REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
The Design and Implementation of a User Interface for the Computer-Aided Prototyping System (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Dixon, Robert Mobley

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis		1992, September, 24	289

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Rapid Prototyping, Software Engineering, PSDL, CAPS, Embedded Systems, XWindows, Motif

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Computer Aided Prototyping System (CAPS) is a software tool intended to make the software development cycle more efficient. Through the use of prototyping, the functional requirements for an embedded system can be quickly validated before wasting resources on an inadequate design. As CAPS is composed of separate tools, the user interface must tie these separate programs together into one coherent tool easily used by a software engineer. This thesis discusses the design and implementation considerations involved in creating an improved user interface for CAPS. The new interface is simpler for the user to use and configure. It is also easier for the system administrator to make changes as tools are added to the system and improved. The new interface also tightly couples the Prototype System Description Language (PSDL) syntax-directed editor to the graph editor, and enables automatic propagation of constraints between the two. The final design is presented, along with the implementation. The thesis also contains a manual for using the system as well as a programmer's manual.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Luqi	22b. TELEPHONE (Include Area Code): (408) 646-2174	22c. OFFICE SYMBOL: CSLq

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

T260090

Approved for public release; distribution is unlimited

The Design and Implementation of a User Interface for the Computer-Aided Prototyping System

by
Robert Mobley Dixon
Captain, United States Marine Corps
B. S., United States Naval Academy, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

24 September 1992

ABSTRACT

The Computer Aided Prototyping System (CAPS) is a software tool intended to make the software development cycle more efficient. Through the use of prototyping, the functional requirements for an embedded system can be quickly validated before wasting resources on an inadequate design. As CAPS is composed of separate tools, the user interface must tie these separate programs together into one coherent tool easily used by a software engineer. This thesis discusses the design and implementation considerations involved in creating an improved user interface for CAPS. The new interface is simpler for the user to use and configure. It is also easier for the system administrator to make changes as tools are added to the system and improved. The new interface also tightly couples the Prototype System Description Language (PSDL) syntax-directed editor to the graph editor, and enables automatic propagation of constraints between the two. The final design is presented, along with the implementation. The thesis also contains a manual for using the system as well as a programmer's manual.

Thesis
D61232
c.1

THESIS DISCLAIMER

Ada is a registered trademark of the United States Government, Ada Joint Program Office.

InterViews is a registered trademark of Stanford University.

Macintosh is a registered trademark of Apple Computer, Inc..

Motif is a registered trademark of the Open Software Foundation.

X Window System is a trademark of The Massachusetts Institute of Technology.

TABLE OF CONTENTS

I. INTRODUCTION 1
A. GENERAL 1
B. PROBLEM STATEMENT 2
C. SCOPE 3
II. BACKGROUND 4
A. GENERAL 4
B. PROTOTYPING SYSTEM DESCRIPTION LANGUAGE 4
C. COMPUTER AIDED PROTOTYPING SYSTEM 6
D. EVOLUTION STEPS 7
E. X WINDOW SYSTEM 9
F. INTERVIEWS 10
G. UNIDRAW 11
H. MOTIF 11
III. DESIGN OF THE CAPS '93 USER INTERFACE 12
A. GENERAL 12
1. Nature of the Team 12
2. Nature of the Tools 12
3. Nature of CAPS 12
4. Nature of User Interface Design 12
B. DESIGN PROCESS 13
C. DESIGN RESULTS 14
1. General 14
2. Maintaining Consistency 14
3. CAPS '93 Environment 15
IV. IMPLEMENTATION CONSIDERATIONS 16
A. INTERVIEWS 16
1. Encapsulated Functions 17
2. Lack of Documentation 17
3. Use of Inheritance 18
B. MOTIF 18
C. IMPLEMENTATION CONSIDERATIONS 19
D. MENU IMPLEMENTATION CONSIDERATIONS 19
E. GRAPH EDITOR IMPLEMENTATION CONSIDERATIONS 20
F. IMPLEMENTATION SOURCE CODE 20
V. FUTURE WORK 21
A. DATA STRUCTURE CHANGES 21
1. Reduce the Number of Pointers 21
2. Incompletely Defined Objects 22
3. Time Units 22
4. Display Output 22
5. Spline Generation 23

a. Number of Points 23
b. Performance 23
Appendix A 24
I. CAPS USER INTERFACE DESIGN DOCUMENTATION 24
A. STATEMENT OF PURPOSE 24
B. ASSUMPTIONS 24
C. GOALS/CONSTRAINTS HIERARCHY 25
D. PRELIMINARY DESIGN 28
E. TEST LIST FOR GRAPH EDITOR 33
Appendix B 35
I. USER'S MANUAL 35
A. GENERAL 35
B. INSTALLATION 35
C. CUSTOMIZATION 35
D. STARTING CAPS '93 36
1. Step Menu 37
2. Browse Menu 40
3. Help 41
E. WORKING WITH STEPS 41
1. Editing 43
2. Running a Prototype 44
F. WORKING WITH THE PSDL EDITOR 45
1. General Concepts of the PSDL Editor. 45
2. Interaction Between the Syntax-Directed Editor and the Graph Editor 47
3. Operation of the Graph Editor 48
a. General 48
b. Mouse Features 49
c. Specifying defaults 49
d. Working with Operators and Terminators 49
e. Working with Streams 52
f. Working with Text Strings 53
g. Graph Options 54
4. Operation of the Syntax-Directed Editor 55
Appendix C 56
I. PROGRAMMER'S MANUAL 56
A. ENVIRONMENT 56
B. GENERAL 56
İ. Introduction 56
2. Inheritance 57
3. Object-Oriented Methodology 58
4. Debug Code 60
C. MODULE DESCRIPTION 61
1. Menu shell 61

a. Implementation Details. 61
2. Interface to Syntax-Directed Editor 62
a. Implementation details. 64
3. Graph Editor 65
a. Implementation Details. 66
4. GraphObjectList 69
5. GraphObject 69
a. Implementation Details 70
6. OperatorObject 71
7. StreamObject 72
a. Implementation 72
8. SplineObject 73
a. Implementation 73
9. FontTable 76
Appendix D 77
REFERENCES 276
INITIAL DISTRIBUTION LIST 277

LIST OF FIGURES

Figure 1, Example PSDL Graph 5
Figure 2, Decomposed PSDL Graph 6
Figure 3, Evolution Step Diagram 8
Figure 4, Evolution History Diagram 9
Figure 5, Initial Menu 29
Figure 6, A Step Has Been Selected 30
Figure 7, Syntax-Directed Editor Editing PSDL 31
Figure 8, Graph Editor Editing PSDL 32
Figure 9, CAPS '93 Main Menu 36
Figure 10, Step Menu 37
Figure 11, New Step Submenu 38
Figure 12, Open Step Selection Box 39
Figure 13, Browse Menu 40
Figure 14, Help Menu 41
Figure 15, Save Step Menu 42
Figure 16, Edit Menu 43
Figure 17, Run Menu 44
Figure 18, PSDL Editor in Text Mode 46
Figure 19, PSDL Editor in Graph Mode 48
Figure 20, Draw Options Menu 51
Figure 21, Properties Dialog Box 52
Figure 22, Graph Options Menu 54
Figure 23, Inheritance Tree for Graph Editor 57
Figure 24, Layers of X 58
Figure 25, Graph Editor Structure Chart 67
Figure 26, Shadow Points 74
Figure 27, Constructed Center Point 75

I. INTRODUCTION

A. GENERAL

The breadth and scope of the problems facing software designers and programmers today are well understood in the computer science community and in the nation at large. As hardware becomes more capable, the software that drives the hardware becomes more complex. As software and operating systems become more complex, the human frailties of the supposed masters of these systems become more apparent.

Designers who can't remember their spouse's birthdays are required to know and understand the intricate functioning of software systems with thousands of modules, each with many parameters.

Programmers who can't give adequate directions to lost strangers are asked to communicate the interfaces and interrelationships between modules of source code the dimensions of a telephone directory.

Computer scientists who can't type reliably without looking at the keyboard must correctly create and enter thousands of pages of source code, where the simple substitution of a period for a comma can bring down a system.

Our society depends increasingly on the functioning of software. Everything from microwave ovens to intercontinental telephone systems, video games to jet airplanes are now reliant on the functioning of programs that must be error-free.

The trend in the software engineering field is towards automated tools that handle the tedious mechanics of the design process, allowing the human mind to do what it does best. Machines excel at the repetitive and mundane, but the human mind is capable of creative thought and inference beyond the abilities of the fastest supercomputer. Although computers make good spell-checkers, no computer can compete with the creative writing skills of a third-grade child.

The Computer Aided Prototyping System (CAPS) is a collection of software engineering tools that help the designers of large embedded systems to create complex systems. It allows the human being to concentrate on the high-level details of these systems, while verifying that the system can actually be implemented and operate in the way that the designer desires.

To use CAPS, the user specifies the relationship between software objects, including the data flow between them and the necessary constraints. These specifications are expressed in the Prototype System Description Language (PSDL). The user ideally works at a high level of abstraction, working his/her way down in terms of scope and up in terms of complexity. Eventually an atomic operator is reached that can no longer be subdivided. The user has the option of selecting existing code that performs the desired operation, or actually implementing the operator in a higher-level language such as Ada. At this point, CAPS generates the necessary code to link the operators together, and notes any inconsistencies in the specifications that make it impossible to translate. The prototype is compiled and executed, allowing the user to verify that the system performs according to its specification. CAPS automatically enforces the timing constraints expressed in the PSDL and provide descriptive messages pinpointing timing problems.

The above is an extremely "soft" description of CAPS, but it's adequate to provide an introduction to the system. A more complete description of CAPS and PSDL may be found in Chapter II.

B. PROBLEM STATEMENT

CAPS has been the focus of research efforts for several years. Many different pieces of CAPS have been implemented as the end product of Masters' theses over different periods in time, during different stages of the design effort, using system models with different levels of maturity. As the different tools have been created they have been combined into a somewhat less than coherent whole.

The focus of this thesis is to create a simple, top-down design for the user interface of the 1993 version of CAPS by examining the overall functioning of CAPS and taking into account current models being used for the design process.

C. SCOPE

The scope of this thesis was originally limited to connecting together existing tools. As the thesis process continued, however, it became apparent that some of the original tools weren't up to the job. Although created in accordance with CAPS design as it existed at that time, our understanding of the problem had improved as a result of previous work to a point where improvements needed to be made.

This thesis contains three primary products. The first is the design for the CAPS '93 user interface. The second is the implementation of a drawing editor for the graphic representation of the prototype. The third is the implementation of a menu shell for the future CAPS tools.

II. BACKGROUND

A. GENERAL

The Computer Aided Prototyping System (CAPS) is a software engineering tool designed to take specifications expressed in the Prototype System Description Language (PSDL) and make them executable. The current design of CAPS allows the user to work with two separate views of the prototype: a text-based view and a graphics view. It is desirable to view prototype creation as a network of evolution steps.

One constraint on the graphics view is that it must be implemented in the X Window System, from the Massachusetts Institute of Technology (MIT). This constraint is motivated by portability concerns. The simplest way to implement an X Windows-based graphic tool is to use one of the existing toolkits.

During the process of this thesis three toolkits were evaluated: the InterViews toolkit from Stanford, the Unidraw toolkit from Stanford, and the Motif toolkit from the Open Software Foundation. This section provides a general introduction to these issues, as well as references for further study.

B. PROTOTYPING SYSTEM DESCRIPTION LANGUAGE

The Prototyping System Description Language (PSDL) is a text-based language designed to express the specifications of real-time systems. It is based on a graphic model of vertices and edges, where the vertices represent *operators*, or software processes, and the edges represent the conceptual "flow" of data from one operator to another. Each vertex and edge may have associated timing constraints, and the vertices may have associated control constraints.

Formally, the model used is that of an augmented graph,

$G = (V, E, T(v), C(v))$

where *G* is the graph, *V* is the set of vertices, *E* is the set of edges, *T(v)* represents the timing constraints for the vertices, and *C(v)* represents the control constraints for the vertices [LUQI88-2, p. 1411].

Conceptually, PSDL operators may contain other operators to support the principle of abstraction. Effectively, the prototype may be expressed as a *flat* graph, or a one-level graph containing all the *atomic* operators and their streams. An atomic operator is one that is implemented in a programming language, vice a *composite* operator consisting of other operators and streams.

For example, consider the diagram of the following PSDL prototype:

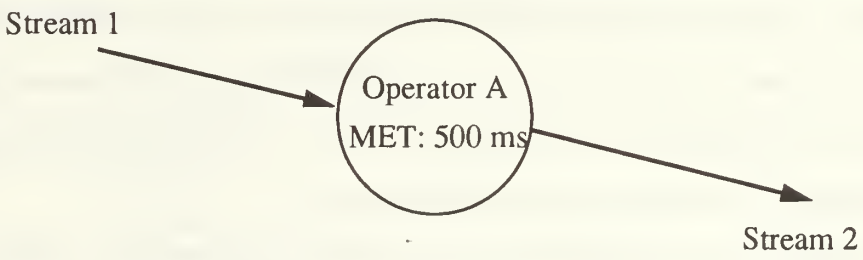

Figure 1. Example PSDL Graph

This represents an operation modelled by Operator A that accepts one data item from Stream 1, performs some operation on the data, and outputs Stream 2. The Maximum Execution Time (MET), or the maximum time the operator may take to execute, is defined as 500 milliseconds.

In this example, Operator A is decomposed as follows:

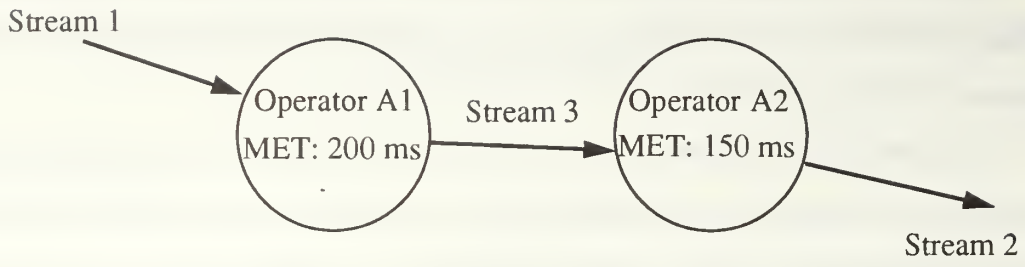

Figure 2. Decomposed PSDL Graph

Operator A is a composite operator, while Operator A1 and Operator A2 are atomic operators, implemented in Ada or some other language. Note that the timing and control constraints on Operator A1 and Operator A2 must be consistent with those of their parent operator. METs for the child operators that summed to a value greater than that of their parent would represent a semantic error in a single-processor execution environment.

Note also that Operator A is not really necessary to implement the prototype. It serves merely to abstract the functioning of its child operators.

Using PSDL, we could express the above example as a system with a single input and a single output, with a maximum execution time of 500 ms. We can further express that the system contains two subordinate process with single inputs and outputs, with maximum execution times of 200 ms and 150 ms respectively. We can also specify the order in which the processes must execute for the system to operate correctly.

A more detailed and rigorous description of PSDL may be found in [LUQI88-2] and [LUQI89-1].

C. COMPUTER AIDED PROTOTYPING SYSTEM

From the introduction to PSDL in the previous section, it is obvious that a method for actually executing PSDL prototypes is very desirable. The software designer would be able to see whether the PSDL prototype was logically consistent and semantically and syntactically correct. It would also allow the designer to verify whether the assigned

constraints were achievable, or whether either looser constraints or a more efficient implementation were required.

The Computer Aided Prototyping System (CAPS) was created with these goals in mind [LUQI88-1]. It is designed to provide the following features to the user:

- automated support for PSDL text editing
- a drawing editor for PSDL graphs
- generation of schedules and control code
- a run-time environment to monitor the execution of prototypes, flagging timing problems and other constraint violations
- a data base of reusable components for implementing atomic operators
- automated support for editing and compiling source code for atomic operators
- a sophisticated engineering data base for managing prototypes

CAPS currently consists of the following tools:

- Syntax-directed editor
- Graph Editor
- Expander
- Translator
- Design data base
- Software base
- Static scheduler
- Dynamic scheduler
- Ada source code editor
- Ada compiler

The specific requirements for the CAPS user interface is discussed in Chapter III. A good introduction into the functioning of CAPS may be found in [LUQI92].

D. EVOLUTION STEPS

Software rarely emerges from the mind of the designer fully formed. It is more commonly formed and crafted gradually, much like a sculptor works on a raw block of stone. He/she patiently chips and chisels a piece at a time until the creation is complete.

A designer first tries to decide what the code is supposed to do, then implements a rough approximation. After a cyclical process of design review and prototyping, coding and debugging, the module is linked into place.

One major difference between sculpture and coding is that the programmer can go back to previous work and try different ways of accomplishing the same thing. The programmer can also abandon changes that once looked promising but didn't work out to be the best formulation.

[LUQI89-2] introduces a graphical representation of the evolution of a software model that clearly describes the change history of a particular module, as well as providing a more formal description of the change process. The revision of a software module can be described graphically in the following way:

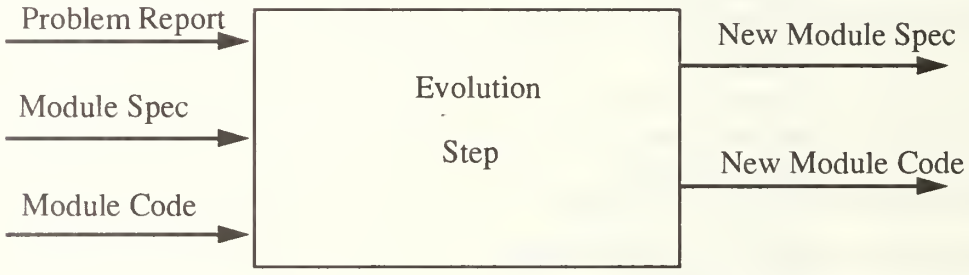

Figure 3. Evolution Step Diagram

The evolution history of a particular module can also be graphically described.

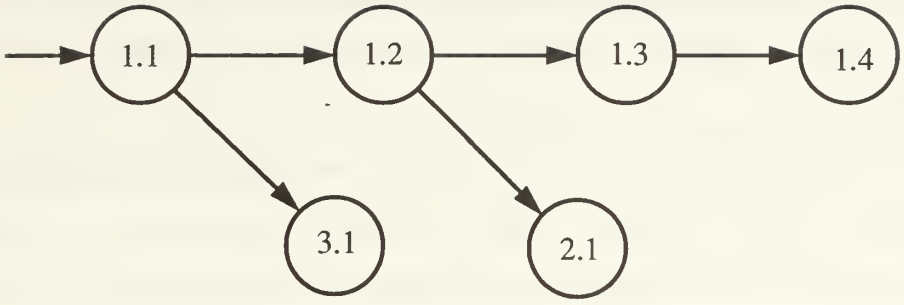

Figure 4. Evolution History Diagram

In the above diagram, versions 1.1 through 1.4 were created in that order. At this point, the designer decided to make changes to a previous version, which resulted in version 2.1. Still searching for the best implementation, the original version was modified, leading to version 3.1.

Although the example focused on the efforts of a single designer, this model is well suited to describing the activities of a design team. A senior designer may direct the evolution of the entire project using the top level of the graph, while subordinate teams work at a lower level, on the subcomponents of the versions. The model also supports a parallel approach, where teams work on alternate solutions to the same problem.

[LUQI89-2] is a more thorough discussion of evolution steps.

E. X WINDOW SYSTEM

The X Window System was developed at the Massachusetts Institute of Technology to solve a problem common in large computer networks: the need to display graphics on many different hardware platforms. The solution was to implement a client-server model a little different than that used to describe computer networks.

In the X Window System, a program is logically separated from the physical display that shows its output. The program is called a *client*, while the physical display unit is called

a *server*. The client sends the server packets of information that specifically describe what the server is to display. The server then executes the desired display instructions. Using X, the client can then send the same display instructions to any server it has access to, which then uses a hardware-dependent method of executing the server's instructions [JOHNSON89].

The graphics capabilities of X are often used to create graphical user interfaces for Unix workstations. These interfaces support the point-and-click style of interface developed at the Xerox, Inc. Palo Alto Research Center (PARC), popularized by the Apple Macintosh, and imitated by Microsoft Corporation on IBM personal computers and compatibles.

The basic X commands are implemented in reasonably low-level graphics primitives, namely lines, circles, ellipses, rectangles, etc.. The need to implement higher-level interface objects to represent things like pushbuttons and scrollbars soon became apparent, as well as the need to simplify the programming model using abstraction and encapsulation. This led to the appearance of a number of X Windows-based toolkits.

F. INTERVIEWS

InterViews is a toolkit developed in C++ at Stanford University [STANFORD91]. It provides a number of predefined objects, including low-level objects such as pushbuttons and scrollbars as well as more sophisticated objects like file management dialog boxes and text editors. Version 3.0 of InterViews also comes with a sophisticated interface generation tool called ibuild. Using ibuild, the programmer can lay out anything from simple text dialog boxes to complete drawing editors. ibuild generates the source code to create the object on-screen, allowing the programmer to wire in his/her application-specific logic.

InterViews makes extensive use of C++ features such as inheritance, making it a rather educational example of object-oriented programming and the C++ language.

G. UNIDRAW

Taking a further step, John Vlissides at Stanford University used the InterViews toolkit to create a higher-level toolkit designed for the creation of domain-specific graphic editors [VLISSIDES89]. This toolkit is called Unidraw. Using a standard set of atomic and composite tools, Unidraw is capable of creating an extremely complex graphic editor with a minimum of source code.

H. MOTIF

Another toolkit available for designing X user interfaces is the Open Software Foundation's Motif toolkit. The Open Software Foundation (OSF) is a consortium of companies including Hewlett-Packard, Digital Equipment Corporation, IBM, and others, whose mission is to enhance the interoperability between computers of different manufacturers.

Motif is a lower-level toolkit than Interviews, providing user interface objects known as *widgets* with a specific appearance and function. The programming environment provided by Motif is commonly a mix between native X, an intermediate level known as Xt Intrinsics, and higher-level Motif. The programmer can thus use the pre-defined Motif objects to provide a user interface with a highly standardized appearance, while at the same time having access to low-level X functions to implement more specific control over the application.

The Motif environment is highly dependent on external configuration options known as *resources* that allow the installed site and the user a great deal of control over the appearance and functioning of Motif programs. If the programmer so desires, almost every aspect of a Motif program from the color to menu options to fonts can be configured in resource files, allowing the users extreme flexibility in the appearance and function of a program. Resources are modified by editing simple text strings, without the need to recompile the original source code [HELLER91].

III. DESIGN OF THE CAPS '93 USER INTERFACE

A. GENERAL

As mentioned in the introduction, the focus of this thesis is the user interface for CAPS '93. Although this was intended to be a complete redesign, without regard to current tools and their interface requirements, there were several factors that tended to complicate the design.

1. Nature of the Team

CAPS is very much a group effort. Many thesis students and many professors and instructors have been involved with CAPS since its inception, and there are divergent opinions on many aspects of CAPS. Completion of this thesis was a sometimes painful process of sorting and sifting these opinions into something approaching a unified design.

2. Nature of the Tools

Several of the CAPS tools are implemented using toolkits, each of which has its own set of features and limitations. It was not initially clear exactly how much flexibility the tools would offer in actually implementing the design.

3. Nature of CAPS

CAPS exists to automate the creation and execution of PSDL prototypes. PSDL is a text language with a number of syntax options that are often confusing to new users. The resulting interface must support the full set of PSDL syntax, while at the same time being useful to new users who aren't PSDL experts.

4. Nature of User Interface Design

The primary objective of user interface design is to create an interface that will be intuitive to the user. Although simple in concept, this is often difficult in execution. There are as many different definitions of "intuitive" as there are users. Many users with good

memories and much experience prefer command-line style interfaces that allow them to immediately enter and execute the commands they want. Other users with weaker memories and a lesser degree of experience often prefer a more menu-driven interface that suggests possible choices. Still other users prefer a fully graphical object-based interface that allows them to point and click at icons.

B. DESIGN PROCESS

The first step in the design process was to create a Statement of Purpose for the interface, found in Appendix A. A number of assumptions were made about the target user of CAPS, also documented in Appendix A.

Since CAPS is essentially a collection of separate tools, it was also necessary to determine the exact environment under which they would run, and the method they would use to communicate.

Another consideration was the need for consistency with existing applications, balanced against the theoretical models used in CAPS. Most computer users today are familiar with the point-and-click style of interface often implemented under X, as discussed in Chapter II. Computer users generally understand how to use a computer mouse as a pointing device and how to work with pull-down menus.

All the previously mentioned interfaces allow the user to work with files and to navigate around in subdirectories. They usually implement at least two standard items in a pull-down menu: File and Edit. Users have come to expect this.

One of the decisions made in the design and implementation of CAPS was to use the model of the evolution step as discussed in Chapter II. Although this compromises consistency, it allows the user to work with the correct mental model, vice getting bogged down in the directory structure of the user's environment.

The most effective form for documenting these requirements was the goals/constraints hierarchy from [BERZINS91]. The resulting document is contained in Appendix A.

Having arrived at the specifications for the interface, it was necessary to determine its general appearance and structure. A number of different means of evaluating possible interface designs were used, including the use of Macintosh Hypercard stacks, pencil sketches, and Framemaker drawings. A final menu structure was finally approved, contained in Appendix A.

C. DESIGN RESULTS

1. General

The design agreed upon for the main PSDL editing functions is a combination text and graph editing system providing a number of improvements over previous versions of CAPS. Instead of using a generic text editor, PSDL text is edited using a syntax-directed editor generated by the Cornell Synthesizer-Generator. The new editor has the semantics and syntax of PSDL built-in, preventing the user from making most mistakes. It is also able to provide the user with the legal syntax options available at all stages of editing. The correctness of the entered code is improved, as well as simplifying PSDL entry for novices.

The graph portion of PSDL is handled in two modes. In the view mode, the syntax-directed editor is the primary editor, with a graph viewer located alongside, displaying the graph of the current PSDL operator. As the user moves through the PSDL text, the graph automatically updates to reflect the PSDL being viewed. In edit mode, the syntax-directed editor is replaced by a full-function graph editor, capable of adding and deleting operators and streams, editing names and constraints, and modifying graphical attributes such as color, font, and location. When the graph editor terminates, the syntax-directed editor automatically updates the edited prototype with the new graph.

2. Maintaining Consistency

One major problem with the previous versions of CAPS is the issue of consistency between the text and graphic views. It is entirely possible to use the text editor to modify graphical attributes of the prototype. These changes did not automatically reflect in the graph, and the two views become inconsistent.

This is no longer a problem with CAPS '93, as both editors edit the same data structure. Changes to the structure by the syntax-directed editor are always followed by calls that update the graphical display. The graph editor can only be invoked from the syntax-directed editor, so it is possible to check the data structure when the graph editor terminates, and make the appropriate changes to the PSDL attribute tree.

3. CAPS '93 Environment

Previous versions of CAPS relied on a set of environment variables to point to the locations of the directories used while editing the prototype. Under the new design, a ~/.caps directory is created automatically which contains a subdirectory for each prototype. Within each prototype subdirectory, a subdirectory is automatically created and deleted for each version as required.

With the CAPS files in a standard location, the user longer has to configure his/her environment to use the tool. The only requirement is that the CAPS binary is in his/her path, and the menu shell does the rest. The CAPS system is responsible for creating and maintaining the directory and file structures it uses, and the CAPS user is insulated from these details.

IV. IMPLEMENTATION CONSIDERATIONS

A. INTERVIEWS

The previous version of the graph editor and of the menu shell [CUMMINGS90] was written using Stanford's InterViews toolkit, discussed in Chapter II. This version of CAPS was oriented around the graph editor, using the vi editor to edit PSDL source code. InterViews proved to be an excellent choice for this project because an InterViews drawing editor called idraw already existed. This editor was very close in functionality and appearance to the desired CAPS graph editor, and it was possible to modify idraw to achieve the desired results. The initial version of the graph editor was created before there was a syntax-directed editor for PSDL, and hence was designed in isolation.

The CAPS '93 graph editor has a much different set of responsibilities from the previous version. Rectangular operators have been added to represent Yourdon-style terminators [YOURDON89], and concentric circles represent composite operators. Streams with state variables are represented using a heavier line. Perhaps the biggest difference, however, is the requirement that the graph editor be able to interface with the syntax-directed editor in two ways: when the syntax-directed editor desires the graph updated, and when the graph editor is invoked.

Since the previous version of InterViews, Stanford has also introduced the Unidraw toolkit for creating domain-specific drawing editors. Unidraw more completely encapsulates the low-level functioning of the editor.

A one-quarter directed study was conducted to examine InterViews and its features, especially those related to object-oriented programming. Two weeks were spent researching Unidraw and attempting to determine whether Unidraw was a good fit to the problem at hand.

At the end of the evaluation period, it was determined that neither InterViews nor Unidraw would be suitable for the CAPS '93 graph editor. This decision was made for the following reasons.

1. Encapsulated Functions

Both Interviews and Unidraw are designed to encapsulate the low-level graphic functions of X. They both do an excellent job of this, implementing a rich set of graphic interface objects and predefined window types. InterViews and Unidraw are very good at solving the problems they were designed for, assuming the design decisions made for the user are the correct ones. In a situation where the programmer wants more control over the implementation, both Unidraw and InterViews become more difficult to use. The programmer must determine exactly how the object performs the function desired, and then must determine how to modify the tool's default behavior to suit his/her needs.

The CAPS '93 graph editor implements some very specific behavior that is not directly available under InterViews. All functions are performed with the same mouse button in order to simplify the interface. It communicates with the syntax-directed editor by using special X events which require the editor to react. It implements curved lines that always terminate in arrowheads, and always remain connected to the operators they are attached to. It automatically breaks the text strings assigned to operators into multiple lines. It associates a text string with a particular graphic object, so that when the string is selected its parent object displays a visual cue to the user. It also stores the attributes of the graph in a format that can be read and understood by the syntax-directed editor.

2. Lack of Documentation

Both InterViews and Unidraw are research tools developed at Stanford University. As such, the available documentation is limited. Unix manual pages exists for all functions and objects, and a few high-level tutorials exist, but there is little or no documentation that explains how the pieces fit together. There is little reference material for the programmer who wants to modify the behavior of existing tools. The only available

form of documentation in most instances is to examine the uncommented library source code for the tool in hopes of figuring out how things work.

3. Use of Inheritance

Interviews and Unidraw are both C++ toolkits making sophisticated use of *inheritance*. Inheritance is a property of object-oriented languages like C++ that allow the programmer to reuse existing software implementations, adding and modifying features in a structured way. While this is no doubt a convenience for the programmer, it results in some objects being implemented with seven or eight levels of inheritance. At the leaf nodes of the inheritance tree, the reader sees methods and variables used, often with no idea where they are defined. Small pieces of code are added at each level of inheritance, spreading the complete definition of some operations up and down the inheritance tree. It becomes very difficult to figure out what things do, where they do them, and how to modify them to achieve desired results.

B. MOTIF

Motif is a fairly low-level toolkit that requires a mixture of high-level and low-level programming to implement most things. While this becomes tedious in some cases, Motif programming has the advantage of being straightforward. Most things happen right out in the open, instead of being hidden in an inheritance hierarchy. Motif is also a widely used industry standard. Any good technical computer bookstore has a number of titles on various aspects of Motif programming, providing a wide selection of examples to work from. Motif provided the flexibility needed for the approved design while abstracting out many gory details. It took a little over a week from first investigating Motif to producing a simple editor capable of displaying operators and streams.

C. IMPLEMENTATION CONSIDERATIONS

Having chosen a toolkit, it was also necessary to choose a language to implement the graph editor and menu shell in. Since Motif is primarily a C-based toolkit, both C and C++ were options. Bindings also exist for Ada.

Ada was rejected for two reasons: lack of documentation and implementation difficulty. Few commercial books are available on the subject of writing Motif code in Ada. Also, Motif callback functions cause some problems.

In Motif, each interface object must have *callback functions* defined by the user in order to implement desired behavior. A callback function is a normal C function with specified formal parameters. A special function is called to associate the callback with the correct interface object. This function takes as a parameter the address of the function, which enables the window manager to call it.

This presents a problem in Ada, because there is no direct way to get the address of a function. The Motif Ada bindings get around this problem by instantiating a generic package with the callback function as a parameter. This is less straightforward than the C implementation.

C++ was selected over C because of its strong typing, and also because its object-oriented features are a good match for graphics-based programs that manipulate graphic objects. It is conceptually straightforward to implement a graphic object as a C++ object that contains information about its appearance and about how it should respond to various stimuli, such as having its handles and text strings moved.

D. MENU IMPLEMENTATION CONSIDERATIONS

The diagrams for the menu shell initially implemented submenus of pulldown menu items as dialog boxes with option buttons. Motif allows submenus to be implemented as pullright menus. Since this provides a simpler and cleaner interface, Motif pullright menus were used instead of dialog boxes.

E. GRAPH EDITOR IMPLEMENTATION CONSIDERATIONS

Additional menu options were added to the original design as a result of user testing. It was originally thought that a user would have no need for a Load option, because the graph editor edits a file determined by the PSDL editor. It was pointed out that it is a good practice to periodically save work in order to recover from large-scale errors in judgement, so this feature was added. An Undelete Operator option was added for the same reason, as was a Quit Without Saving option.

During implementation it was also realized that a MET can only be entered for an operator and a latency can only be entered for a stream. If the object is already selected on the drawing, it can be determined by the object type whether a MET or latency is being specified. One button was thus eliminated, simplifying the interface.

F. IMPLEMENTATION SOURCE CODE

Source code for both the graph editor/viewer and menu shell are located in Appendix D.

V. FUTURE WORK

In the process of completing a thesis project, many of the best ideas come when it is too late to implement them. This thesis is no different. The following ideas are left to posterity.

A. DATA STRUCTURE CHANGES

The initial data structure used by the CAPS '93 Syntax-Directed Editor (SDE) and the graph editor/viewer was created at a time when the actual implementation of these tools and of the communications interface between them was poorly understood. Experience gained in the implementation process leads to the following recommendations.

1. Reduce the Number of Pointers

The current data structure uses pointers to link streams to the operators they originate in and terminate in. Pointers are also used to reference data values used by the SDE, such as maximum execution time (MET) and the is_composite flag. This practice violates the principle of encapsulation as applied to these data structures, and introduces some dangerous dependencies. For example, what happens when a maintenance programmer decides to "clean up" the procedure for freeing memory assigned to a stream node by freeing the target and destination operators? This memory returns to the system heap for reassignment without actually being erased. At some point, some other routine will probably be given this memory in response to a request for additional memory, overwrite it, and cause a crash the next time the operators are referenced. This crash will come at an unpredictable time and place, which will be difficult to trace back to the actual cause of the problem. The current design does not recycle storage for the data structure used in the interface between the graph editor and the syntax-directed editor, and it is unsafe to add such a feature.

It is recommend that unique identifiers of the necessary operators be placed in the data structure, or the actual values in the case of MET and the is_composite flag. Access routines can be written for the operator list that return any kind of information about the desired operator.

2. Incompletely Defined Objects

The current SDE and graph editor/viewer always assume that the user will pass semantically correct and complete information. They also assume that the user will decide to end his/her work session when the prototype is complete and correct, vice wanting to save work in progress that still needs to be finished. This assumption is unrealistic. The data structure should contain a flag similar to "is_incomplete" so that the tools can ignore incompletely defined objects.

3. Time Units

The current data structure assumes that all time values will be specified in milliseconds, although PSDL supports resolution down to the microsecond. It makes more sense for the data structure to either explicitly contain unit information, or to pass data in microseconds and have the tools select the logical units to display the data.

4. Display Output

There are several pieces of information needed to write graphics data to a display window, including the identifier of the display, the addresses of the output widgets, and the desired *graphics context*. The graphics context is a data structure containing specifics about the graphic output, such as foreground and background color and fonts. This data is currently stored in each major program object.

A more object-oriented approach would be to define an object to encapsulate the display. The major program objects would invoke the display object's methods to show themselves on the screen, reducing the number of redundant class variables as well as the effort required to propagate the display information around to the objects in the program.

5. Spline Generation

a. Number of Points

Each spline is composed of a number of subsplines. Each subspline is generated from a sequence of four control points. Since b-splines are continuous in the first and second derivative, the subsplines join together to form a smooth curve. Before generating a spline, though, it is necessary to determine how many points on the display are necessary to draw each subspline.

A relatively crude method of determining the number of display points is now used that depends on the maximum distance between control points. This value is used for all subsplines of the curve, and generally results in sections with too many points. Excess points slow down the program and clutter the graph. A more sophisticated method should be used that determines the correct number of points for each subspline.

One possible method is to determine the x-y coordinates of the first and last display point in each subspline. For control points P_1 through P_4,

$$P_{first} = \frac{1}{6} \cdot P_1 + \frac{4}{6} \cdot P_2 + \frac{1}{6} \cdot P_3 + \frac{0}{6} \cdot P_4$$

$$P_{last} = \frac{0}{6} \cdot P_1 + \frac{1}{6} \cdot P_2 + \frac{4}{6} \cdot P_3 + \frac{1}{6} \cdot P_4$$

If the difference between the x coordinates is larger, this value would be the number of display points to use. This would result in one display point per column of screen pixels. Otherwise, the difference in y coordinates would be used, which would result in one display point per row of pixels.

b. Performance

Every time the screen is redrawn the spline points are recalculated, resulting in a significant performance burden. The actual x-y coordinates of the spline points should be cached in some way, perhaps in an array, and only recalculated when the spline moves on the drawing.

Appendix A

I. CAPS USER INTERFACE DESIGN DOCUMENTATION

A. STATEMENT OF PURPOSE

The purpose of the Computer-Aided Prototyping System (CAPS) User Interface is to integrate the CAPS tools into a simple, easy-to-use environment to facilitate the creation, modification, and maintenance of prototypes.

B. ASSUMPTIONS

Every user interface must be designed using certain assumptions about the target user. This design makes the following assumptions about the CAPS user:

1. The user will have some working knowledge of PSDL, but may not be an expert.
2. CAPS will be used by a broad spectrum of users, ranging from casual student users to professional software engineers.
3. The CAPS user will know how to use a graphical windowing environment. He/she will be able to navigate through a point-and-click style interface without additional instruction on the use of the mouse, pull-down menus, dialogue boxes, etc.. A help facility will be provided covering the meaning of menu items and the use of relevant tools.
4. The CAPS user will know what a data-flow diagram is, and will be able to specify the operations of the desired prototype using this tool.
5. The person implementing the atomic operators will be familiar with the chosen language.

C. GOALS/CONSTRAINTS HIERARCHY

1.0 - Goals for the CAPS User Interface
- 1.1 - Must run on Sun or compatible workstations under Unix because this is the target environment for CAPS.
- 1.2 - Must target a spectrum of users from novice to expert.
  - 1.2.1 - Must provide on-line help for all CAPS functions.
  - 1.2.2 - Should allow graphic and text-based views of the step to allow the user to work with the most intuitive view.
    - 1.2.2.1 - The transition between text and graphic view should be as simple and seamless as possible
  - 1.2.3 - Should provide command-key equivalents for most common point-and-click functions.
- 1.3 - Must be easy to use so that time can be spent working instead of learning the tool.
  - 1.3.1 - Must use a simple directory structure.
    - 1.3.1.1 - Should use a ~/.caps directory to store steps. This makes the subdirectory easy to find for the tools as well as not visible in normal listings.
    - 1.3.1.2 - Each prototype will have its own subdirectory, with the subdirectory name being the prototype name.
      - 1.3.1.2.1 - Each version/variation of a prototype is a separate step, which will be stored in its own subdirectory under the prototype subdirectory. This keeps the steps separated from each other as well as easy to find for the tools.
        - 1.3.1.2.1.1 - The subdirectory name will be the version number (i.e. 1.1 for fishfarm.1.1.). This makes it easier for the tools to find the right step.
        - 1.3.1.2.1.2 -The step subdirectory will contain a <prototype> subdirectory with all the prototypes PSDL and CAPS-generated files.
        - 1.3.1.2.1.3 - The step subdirectory will contain an <ada> subdirectory containing the steps Ada files and Ada library. This library will be linked into the Ada linkers search path.
        - 1.3.1.2.1.4 - The step subdirectory will contain a <bin> subdirectory containing the executable prototype.
        - 1.3.1.2.1.5 - The step subdirectory will contain a <temp> subdirectory containing all the temporary files for the step.
    - 1.3.1.3 - CAPS should create and remove required temporary subdirectories on the fly. This simplifies installation for the user as well as keeping the temporary directories in a standard location for the tools to find.
    - 1.3.1.4 - CAPS binary should be on the user path. This simplifies installation for the user.
    - 1.3.1.5 - A /usr/lib/caps directory or equivalent should contain a tool location file, and possibly the tools themselves.
      - 1.3.1.5.1 - The tool location file points to the location of the tools, allowing the actual tool and data base location to be transparent to the user.

1.3.1.5.2 - Created and maintained by the system administrator. Simplifies CAPS use for the new user.

1.3.1.5.3 - Overridden as required using environment variables for each tool and data base. Allows CAPS developers to use test versions of tools, as well as secondary data bases.

1.3.2 - Must use a graphical, mouse-driven interface for ease of use.

1.3.2.1 - Must run under XWindows to enhance portability.

1.3.3 - Must provide a consistent user interface.

1.3.3.1 - Should use consistent dialog boxes, buttons, menu bars, etc..

1.3.3.2 - Should avoid modal menus, which tend to be confusing to new users. Menu options may be changed in drop-down menus without totally restructuring top-line menu bars.

1.3.4 - User should be able to use the User Interface without reading any instructions.

1.3.4.1 - Should not require the user to create a specified subdirectory structure.

1.3.4.2 - Should not require the user to set any environment variables to start CAPS.

1.3.4.2.1 - The user interface should check for required environment variables and prompt the user to set them if they are not set, or set them automatically if possible.

1.3.5 - Should allow the user to perform the maximum amount of work with the least amount of input.

1.3.5.1 - CAPS will start in an intelligent manner.

1.3.5.1.1 - If no steps are present, prompt the user for a name and start one.

1.3.5.1.2 - If one step is present, start using that step.

1.3.5.1.3 - If more than one is present, present a menu.

1.3.5.1.4 - If a file is present indicating the last step used, use it.

1.3.5.2 - Where one option is consistently more likely than others, i.e. retrieving the latest version of a step from the DDB, it will be the default.

1.3.5.2.1 - Buttons, menu selections, etc., will allow less likely options.

1.3.5.3 - Should only verify program actions of a destructive nature.

1.3.5.3.1 - Should not make the user verify that he/she really wants to use a tool, continue translation, etc..

1.3.5.4 - Menu trees should be as flat as possible.

1.3.5.5 - Normally, unsaved steps will be saved automatically before translation, compilation, or exiting CAPS.

1.3.6 - Must show the user only relevant information.

1.3.6.1 - The user should only see one menu bar at a time.

1.3.6.2 - Interaction between tools should be transparent.

1.3.6.3 - The user should work with lists of steps and versions, without being aware of what subdirectories they are in.

1.3.7 - Must prevent the user from performing illegal or illogical actions.

1.3.7.1 - Inapplicable menu choices should be grayed out.

1.3.8 - Must allow the user to manage tool output.

1.3.8.1 - Textual output from the CAPS tools and the Ada compiler should go into a scrolling window that the user can close or relocate at will. This allows the user to examine output at his/her own convenience.

1.3.8.2 - Error listings should be placed in a saved file to simplify later review.

1.3.9 - Should provide the user consistent feedback about program actions.

1.3.9.1 - Dialog boxes should inform the user when program actions such as translation and compilation begin and end.

1.3.9.2 - Dialog boxes should inform the user about the status of operations.

1.4 - Must enforce consistency and reliability where possible.

1.4.1 - Steps should be stored in one of two places

1.4.1.1 - The design data base holds well formed, public versions of steps/ operators.

1.4.1.2 - The step directories hold work in progress.

1.4.1.3 - When a step is checked out of the design data base for updating, the steps data base entry becomes read-only to prevent conflicting updates.

1.4.1.4 - When a step is committed to the data base, the appropriate working directories are erased to prevent further modification to a committed version.

1.4.2 - There should ideally be one design data base and software base.

1.4.3 - When a step/operator is checked out of the design data base, its whole dependent sub-tree is checked out.

1.4.4 - Tools should save on exit as a default, with the option to quit without saving.

1.4.5 - All Ada code used for the component will be located in the Ada directory of the step. This allows the complete implementation of a component to be committed to the design data base without possibility of unexpected changes.

1.5 - Must allow the user to create steps without leaving the CAPS environment.

1.5.1 - Must integrate the current CAPS tools into one coherent application.

1.5.2 - Must allow the user to generate a user interface for the step. This simplifies the display of prototype execution data.

1.5.2.1 - Should allow the invocation of a tool such as TAE+ using a menu selection.

1.5.2.2 - Should automate the connection between CAPS-generated step code and step interface code as much as possible.

1.5.3 - Must allow the user to invoke an Ada syntax-directed editor within CAPS.

1.6 - Must provide a tool interface for the various CAPS tools.

1.6.1 - Should provide the user with browsers to look over the design data base and software base.

1.6.1.1 - Executed from the top-line menu bar, it should allow viewing without the option of checking out components. This simplifies just looking around while limiting the possibility of unexpected consequences.

1.6.2 - The Tool Interface should be flexible, to allow future changes to the tool structure.

1.6.2.1 - It should created with the expectation that the file structure for components will probably change.

1.7 - Should conform to the required theoretical models.

1.7.1 - Steps should be managed using step names.

1.7.2 - Version numbers will be appended to the component name.

1.7.2.1 - Two digits preceded and separated by a period.

1.7.2.1.1 - First digit corresponds to a variation, or thread.

1.7.2.1.2 - Second digit corresponds to a version of that thread.

1.7.2.1.3 - Each new thread gets a new variation number.

1.7.2.1.3.1 - Numbered sequentially, regardless of parent variation. If there are currently three threads, the next thread off of variation 1.7 is 4.1.

1.7.3 - The user should be able to suspend a step in progress without committing it.

1.7.3.1 - The suspended step will be saved in the design data base in a modifiable state.

1.7.4 - The user should be able to abandon a step in progress without committing it.
1.7.4.1 - The abandoned step will be saved in the design data base in a non-modifiable state.
1.7.5 - The users private workspaces will be read-only to other users.
1.8 - Should provide a separate management/maintenance interface for separate tools, where required.
1.8.1 - A separate management interface allows items to be added and non-step management information to be maintained.
1.8.1.1 - The management interface will allow access to project-related data, such as evolution step hierarchies, programmer assignments, etc..
1.9 - Should be sensitive to the requirement to demonstrate CAPS to non-technical, interested users.
2.0 - Constraints
2.1 - Some required tools are not completely implemented.
2.1.1 - The user will perform all design data base management functions until the management interface is completed.

D. PRELIMINARY DESIGN

The attached diagrams represent the preliminary design of the CAPS User Interface

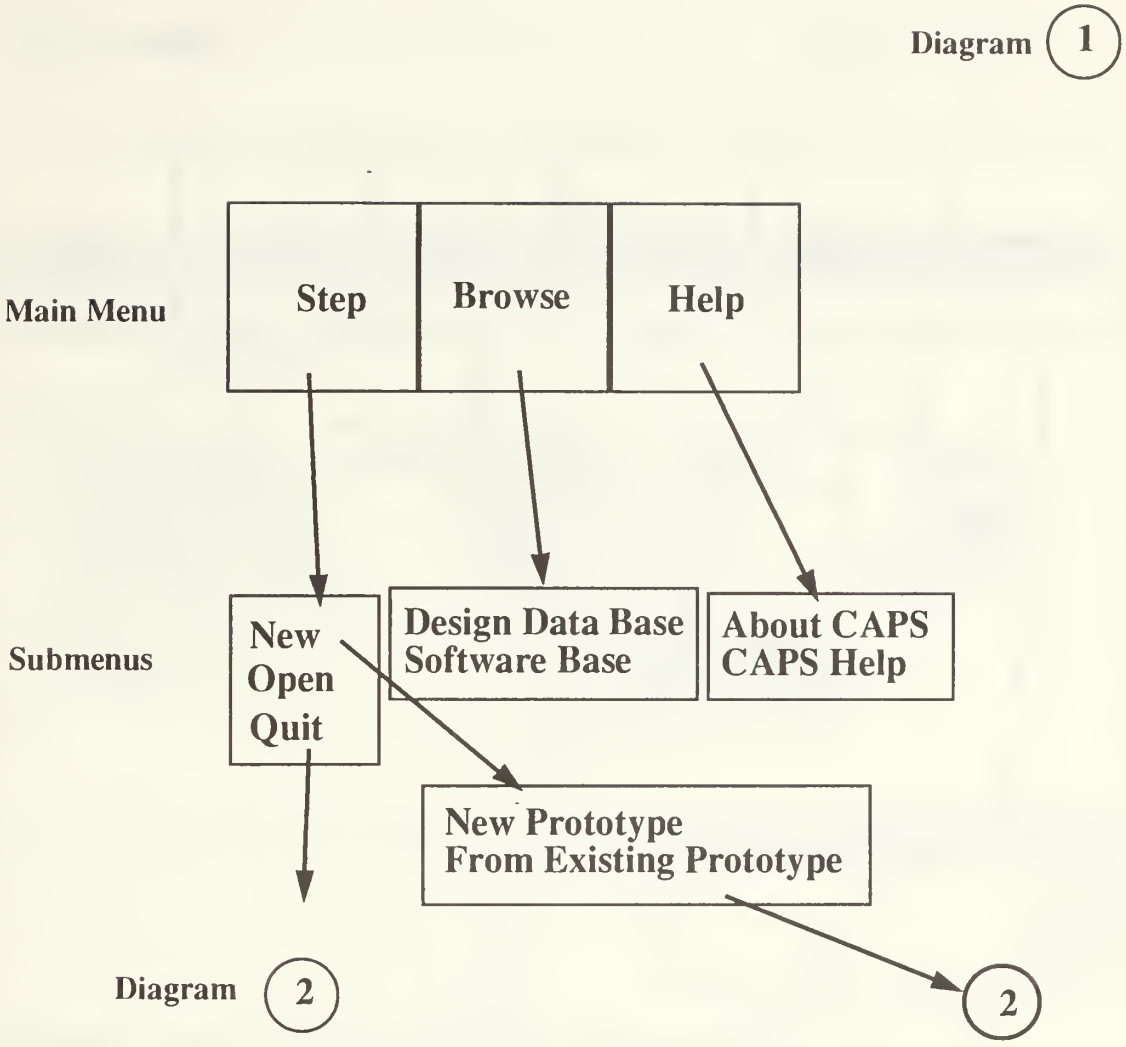

Figure 5. Initial Menu

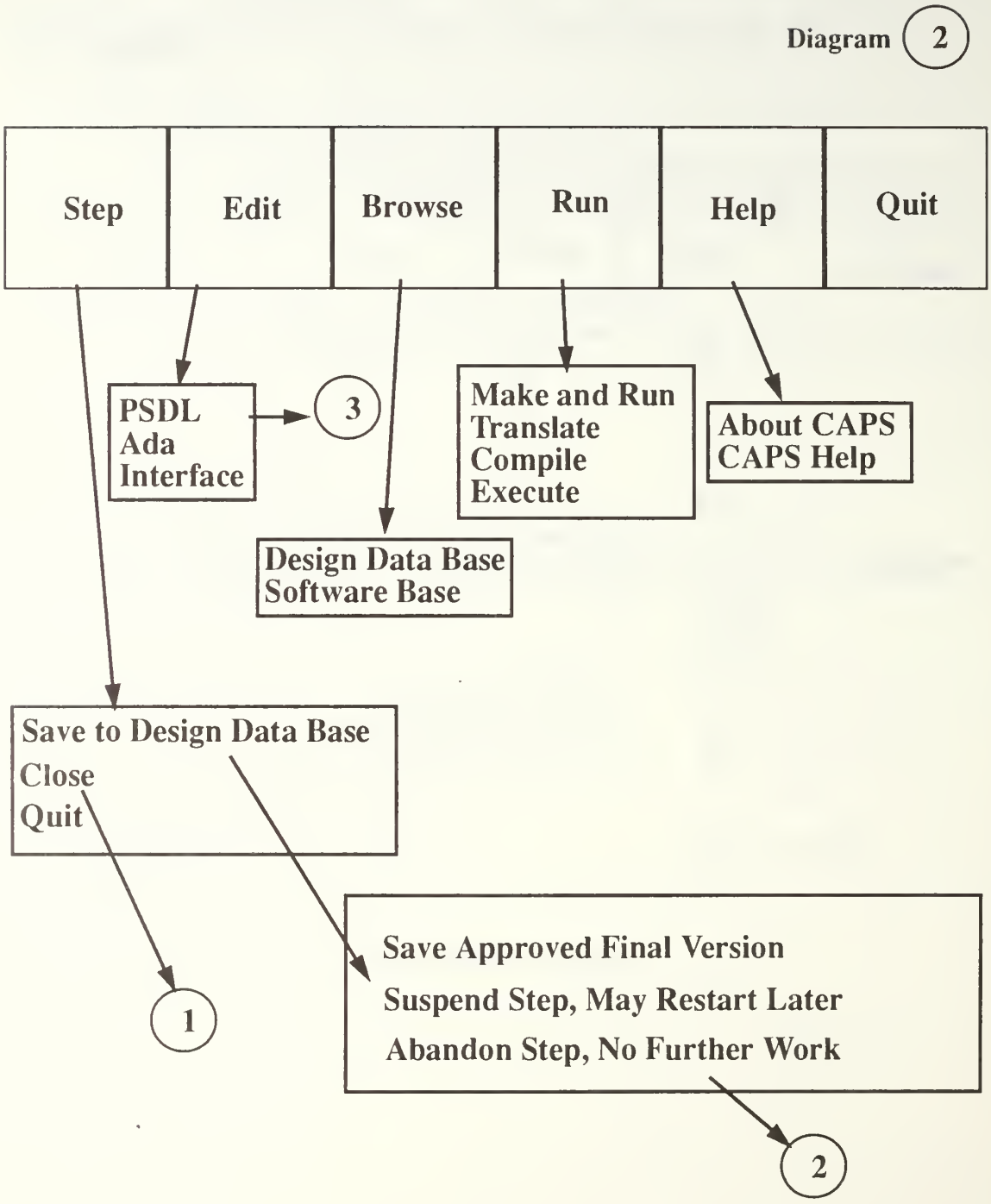

Figure 6. A Step Has Been Selected

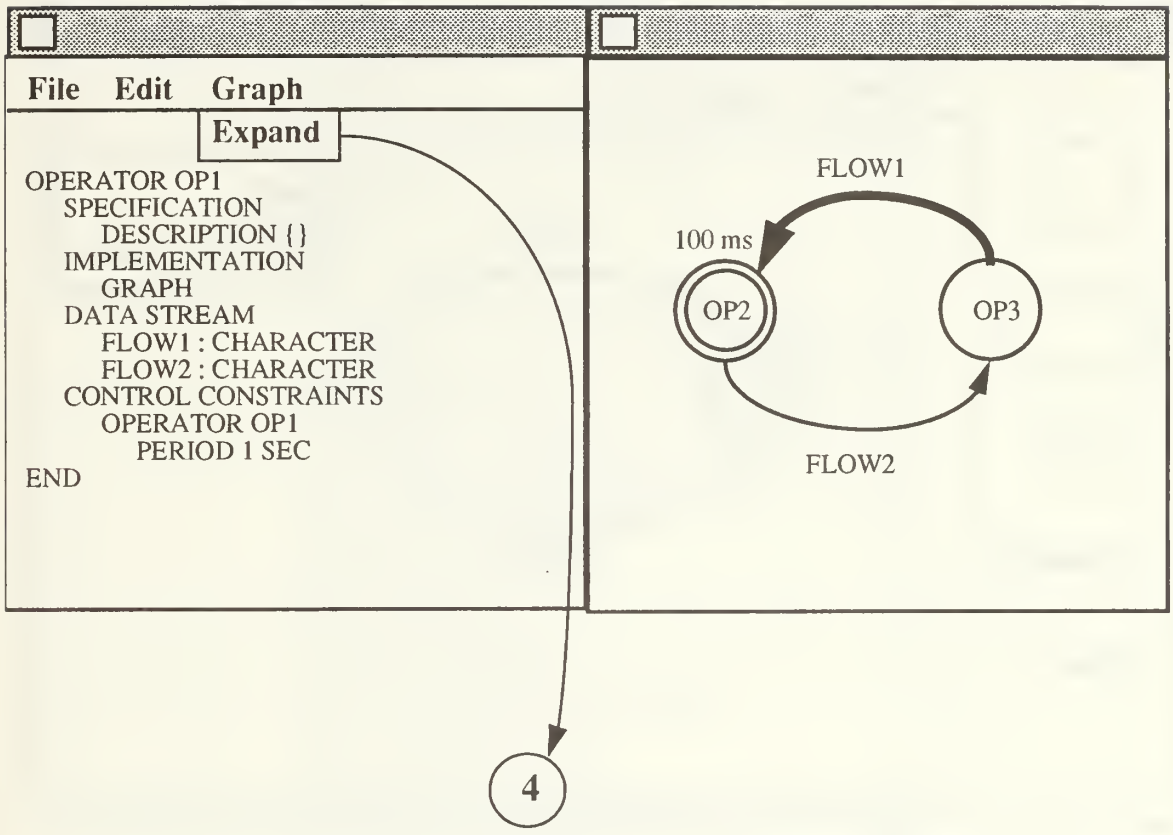

Figure 7. Syntax-Directed Editor Editing PSDL

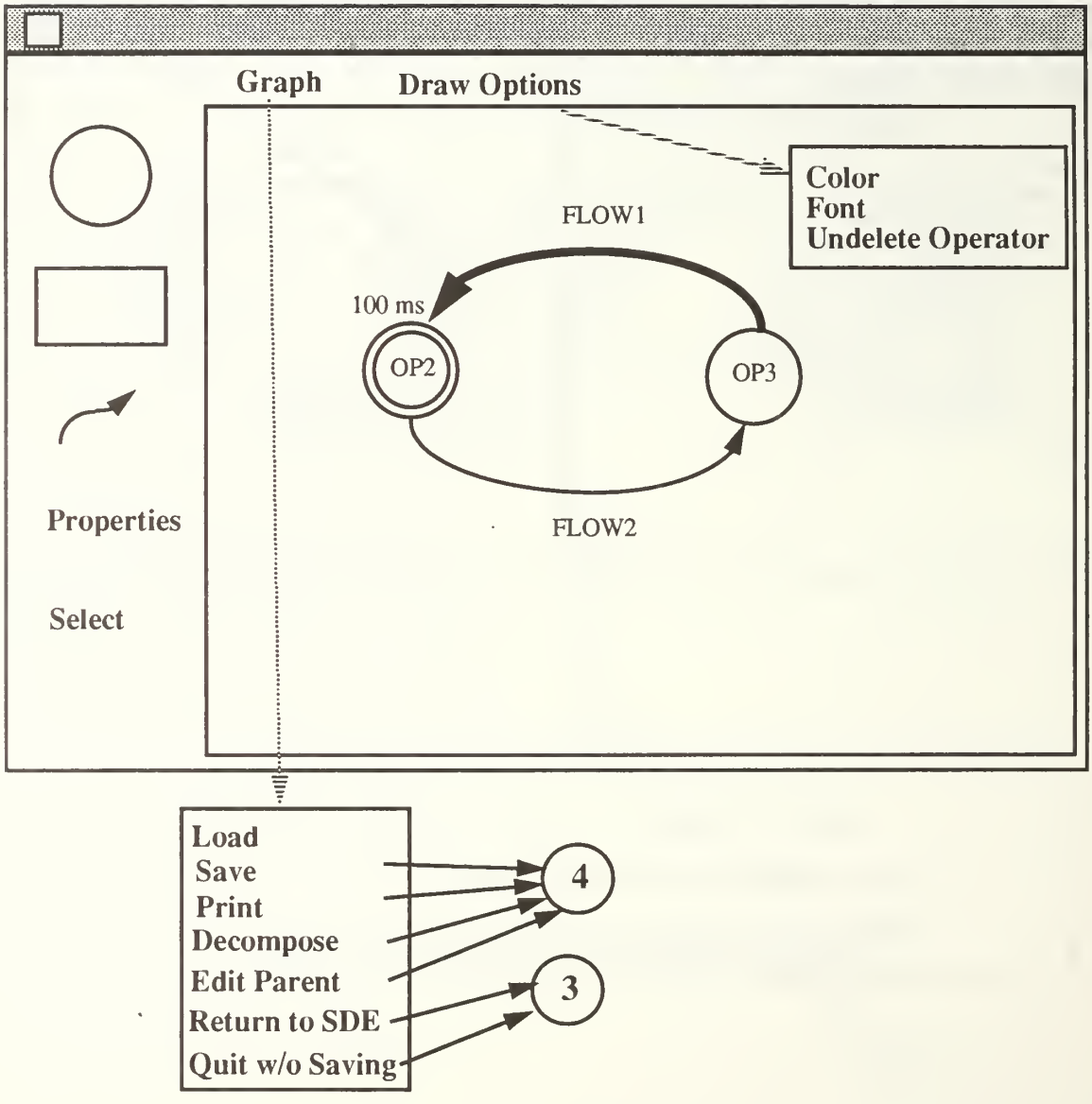

Figure 8. Graph Editor Editing PSDL

E. TEST LIST FOR GRAPH EDITOR

The graph editor must be capable of performing the following test actions:

- Read graph attribute file, new graph if not present
- Set default graph operator/terminator color
- Set default font
- Add operator
- Delete operator
- Undelete operator, attached streams undelete
- Move operator, streams follow
- Add operator name
- Delete operator name
- Move operator name string
- Change operator name font
- Line break on operator name at each underscore
- Add operator MET
- Delete operator MET
- Change operator MET font
- Move operator MET string
- Change operator color
- Add terminator
- Delete terminator
- Undelete terminator
- Move terminator, streams follow
- Add terminator name
- Delete terminator name
- Change terminator name font
- Add terminator MET
- Delete terminator MET
- Change terminator MET font
- Move terminator name string
- Move terminator MET string
- Change terminator color
- Resize operator
- Resize terminator
- Add curved stream
- Add straight stream with no control point, control point added
- Add stream name
- Delete stream name
- Change stream name font
- Add latency

- Delete latency
- Specify state variable stream
- Change latency name font
- Move stream name string
- Move latency string
- Move stream control points
- Add stream with external source
- Add stream with external destination
- Save current graph, continue working
- Restore last saved graph
- Print graph
- Return to SDE, no level change
- Return to SDE, edit parent graph
- Return to SDE, decompose to selected operator
- Return to SDE, abandon graph changes
- Return, indicating decompose to child
- Return, indicating view parent
- Return, indicating no level change
- Return, indicating no update of the data structure
- Update tool indicator with current tool name
- Display graph title in title box
- Use alt-key shortcuts for menu options

Appendix B

I. USER'S MANUAL

A. GENERAL

The CAPS '93 user interface has been influenced by three major issues. First, it is designed around the paradigm of the evolution step, as discussed in [LUQI89-2]. Second, it uses the "select then act" model. Third, since it serves to integrate ongoing thesis work, it is designed for flexibility.

B. INSTALLATION

Installation of CAPS '93 has two parts.

The system administrator's part is to place the tools somewhere in the file structure and to update the /usr/local/caps/bin/tool_location.txt file with the correct locations. This file should also be updated as new tools are installed, and in the event the tools are relocated.

The user's part is to add the subdirectory with the CAPS '93 binary to his/her path. When CAPS '93 is executed it looks for the necessary subdirectories. If it doesn't find them, it makes them.

C. CUSTOMIZATION

Both the caps93 and graph_editor programs have been written to use the default colors and fonts specified in the ~/.Xresources. To change the colors and fonts for these two programs, the following lines should be added if not present, and modified if present:

CAPS_Menu*background: Orange

CAPS_Menu*fontList: -adobe-courier-bold-r-normal--12-20-75-75-m-70-iso8859-1

Graph_editor*background: PaleGreen

Graph_editor*fontList: -adobe-courier-bold-r-normal--12-20-75-75-m-70-iso8859-1

In all cases, the values after the colons should be changed to the desired settings. Often it is necessary to quit X and restart for the changes to take effect.

D. STARTING CAPS '93

Hopefully by now you're ready to take CAPS '93 for a spin. Assuming the executable caps93 file is in your path, just type caps93. After a few moments the main menu comes up.

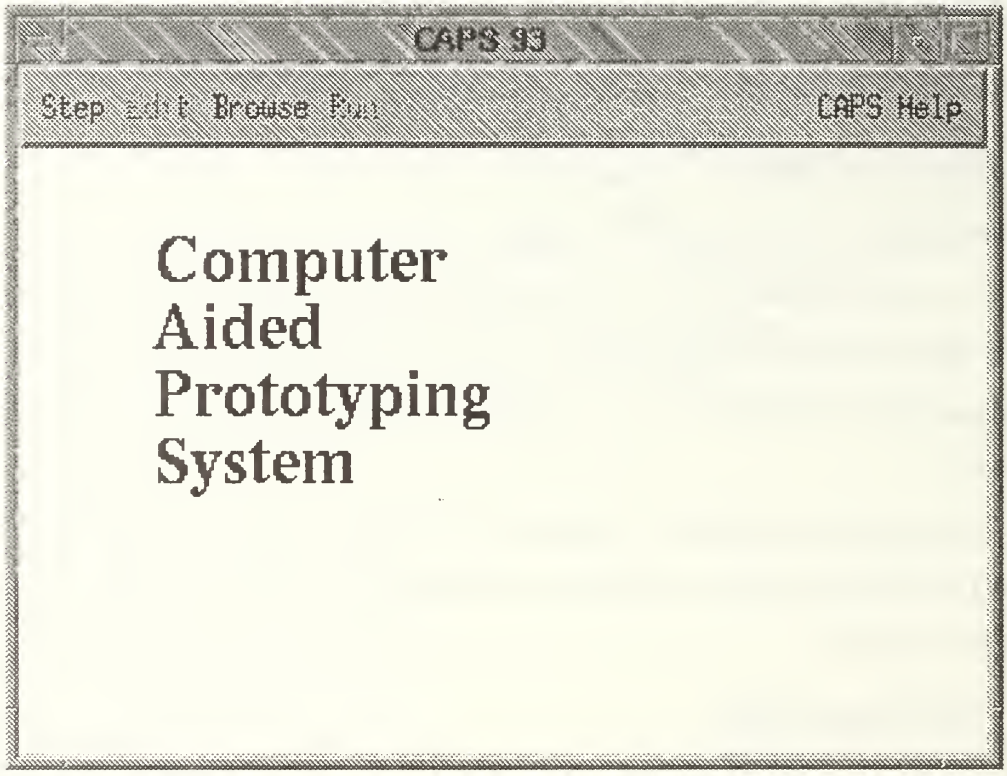

Figure 9. CAPS '93 Main Menu

From this point you have a number of choices. You may select a step to begin working on, or you may browse the Design Data Base (DDB) or the Software Base (SB) to see what's there. You may also use system help to get further guidance on using CAPS 93. Let's discuss these options in detail.

1. Step Menu

As mentioned before, CAPS '93 uses the model of the evolution step. The first thing you must do to begin working is to pick a step to work on, using the Step menu. An evolution step is a means of describing the change history of a program object. Evolution steps are fully described in [LUQI89-2], and in this context mean nothing more than "the object you want to work on."

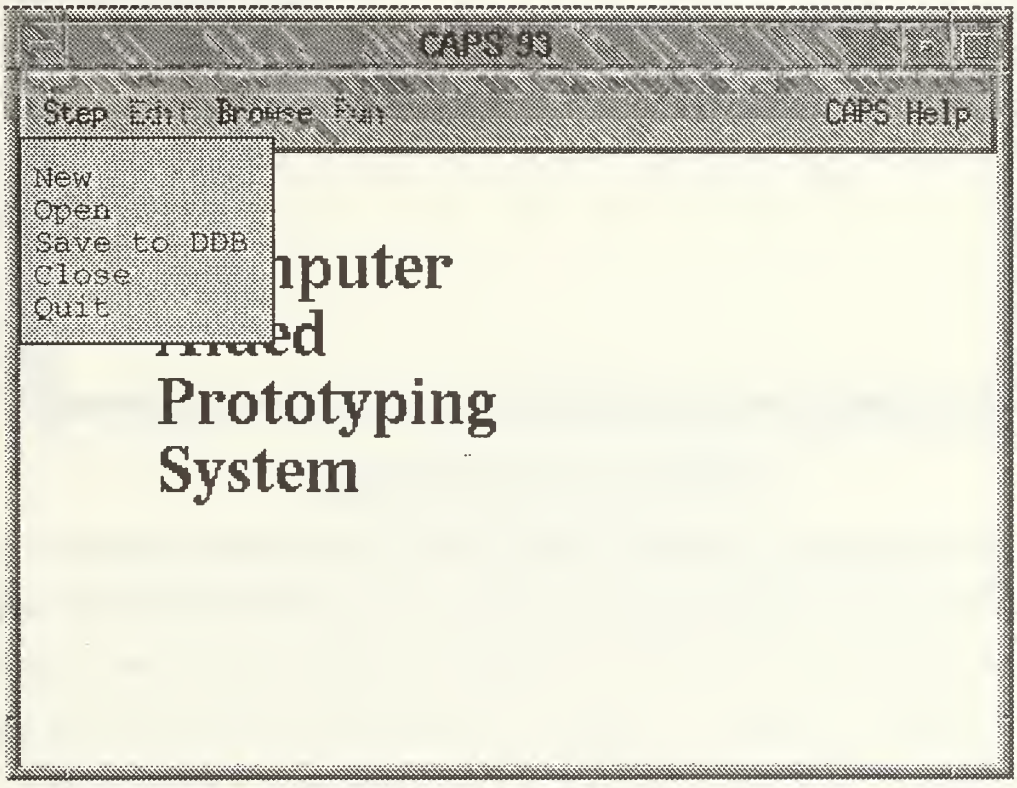

Figure 10. Step Menu

From the Step menu you may either create a new step, or do more work on a previously created step. You may also select Quit to exit the system. Selecting New brings up the New Step submenu.

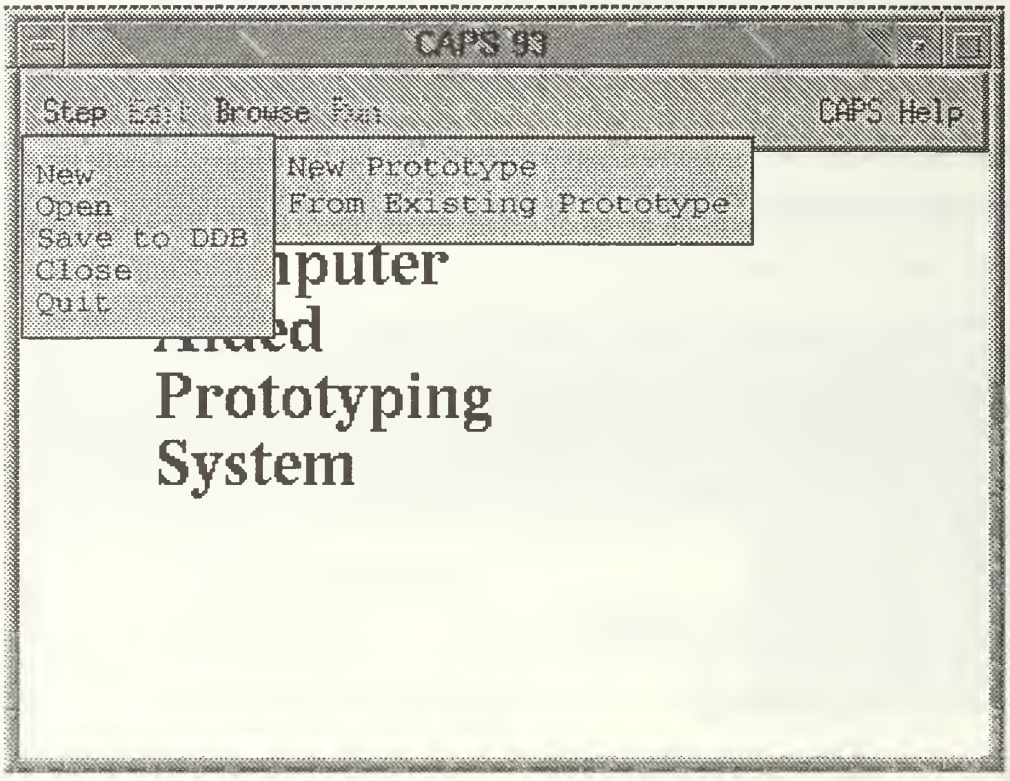

Figure 11. New Step Submenu

New steps may create the first version of a new prototype, or you may create a new version or thread of an existing prototype, from the Design Data Base (DDB). A thread is a new branch in the derivation tree, while a version is a further modification of an existing thread. Whether your change is a version or a new thread is determined by the type of change and the structure of the existing tree. For further information see [LUQI89-2].

To open an existing step, select Open from the Step menu. Opening an existing step allows you to perform more work on a step-in-progress saved in your working area. The user interface provides you with a list of all the steps saved in your work area.

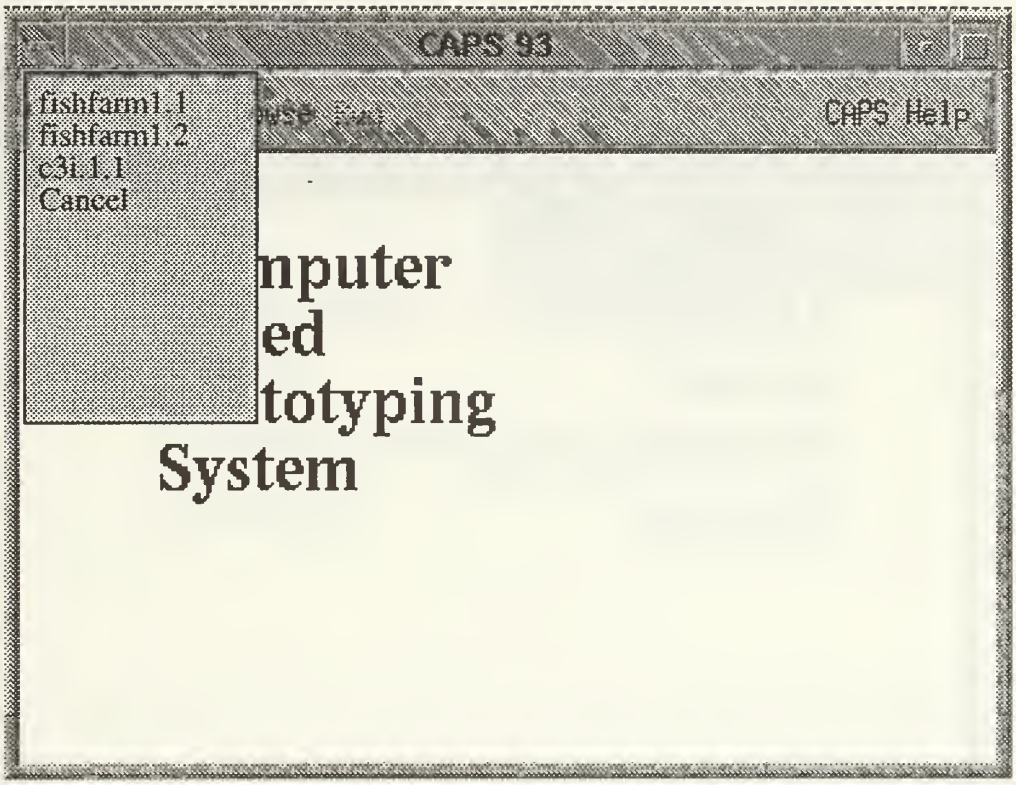

Figure 12. Open Step Selection Box

You may also quit CAPS 93 at any time by selecting the Quit option from the step menu.

2. Browse Menu

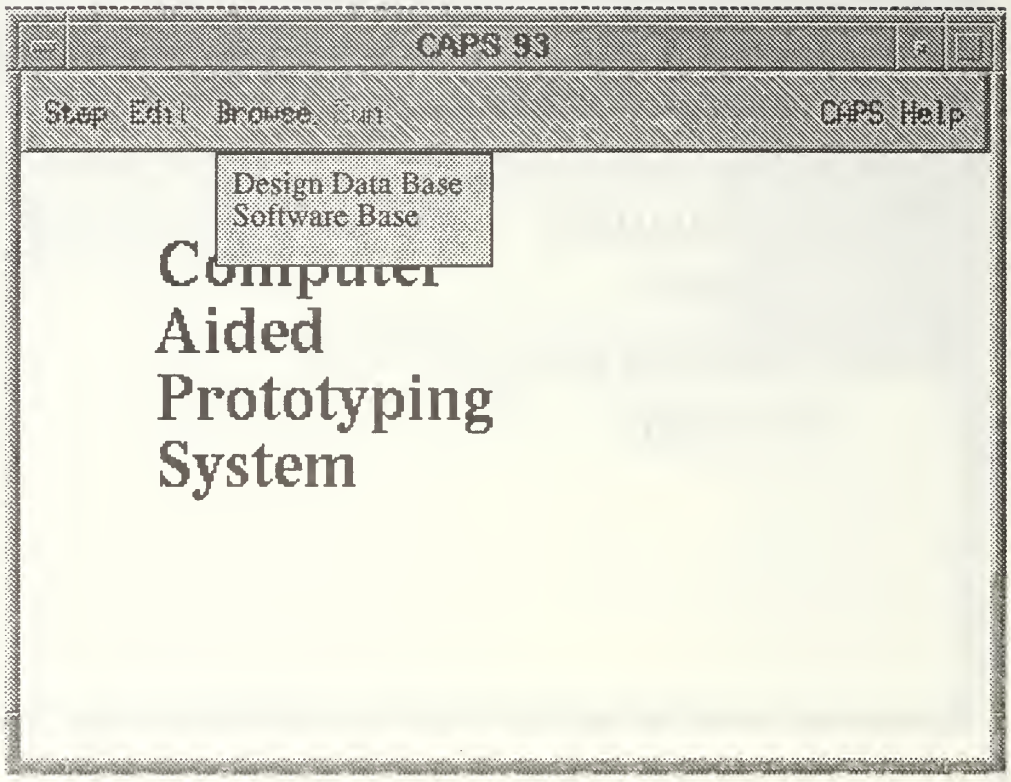

Figure 13. Browse Menu

Another option available when first starting CAPS '93 is to browse either the DDB or the SB. To browse the DDB, select Browse the DDB from the Browse menu. The purpose for doing this is to look at work already saved, to get an idea whether existing work may be modified to meet your needs. While browsing the DDB, all you can do is look, not update. To check out items to work with, use the Step:New menu option to create a new version/variation.

The Software Base is also available for looking, not updating. Browsing the DDB lets you know what PSDL designs are available for reuse, and browsing the Software Base lets you know what pieces of reusable program code are available. Modules from the Software Base are used to implement atomic operators. You may browse the SB by selecting Browse the SB from the Browse menu.

3. Help

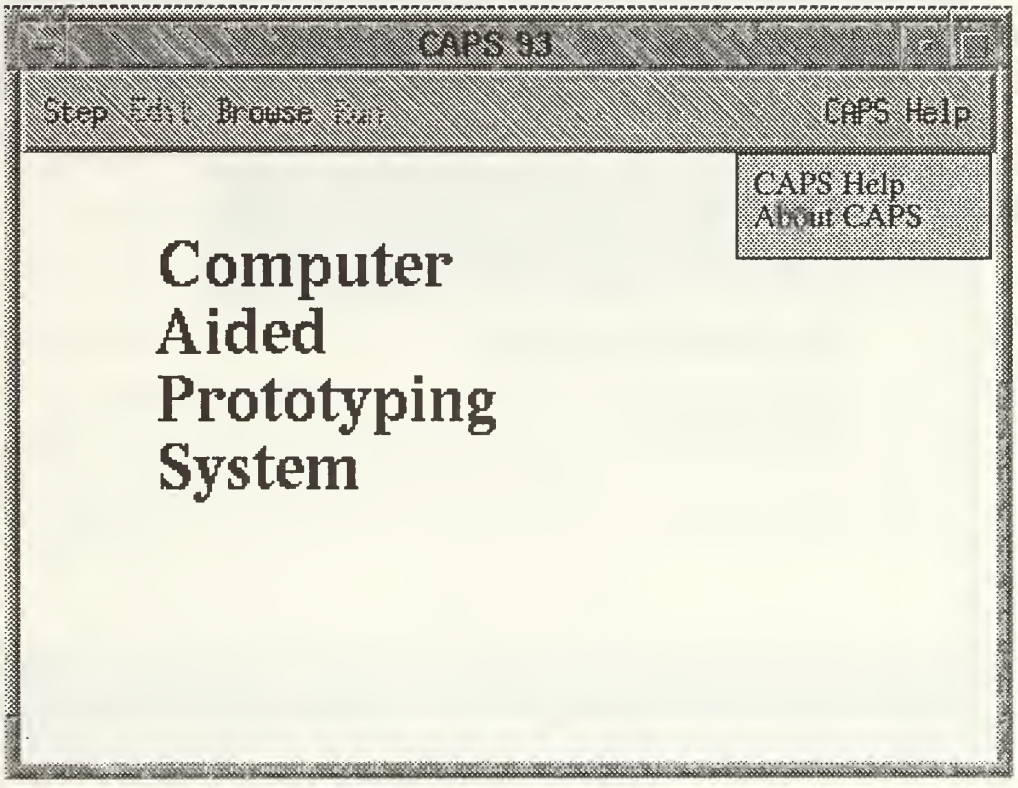

Figure 14. Help Menu

Help is available in two forms. About CAPS tells you a little more about the version of CAPS you're using, where CAPS Help gives you more information about using the tools.

E. WORKING WITH STEPS

Now that you've selected a step to work on, your available menu options from Figure 10 change slightly. You can no longer select a different step because you are already working on one. Since you've selected a step, you have the option of saving the step you're working on to the DDB when the step is complete, or closing it to open and work on another step before you complete the current step.

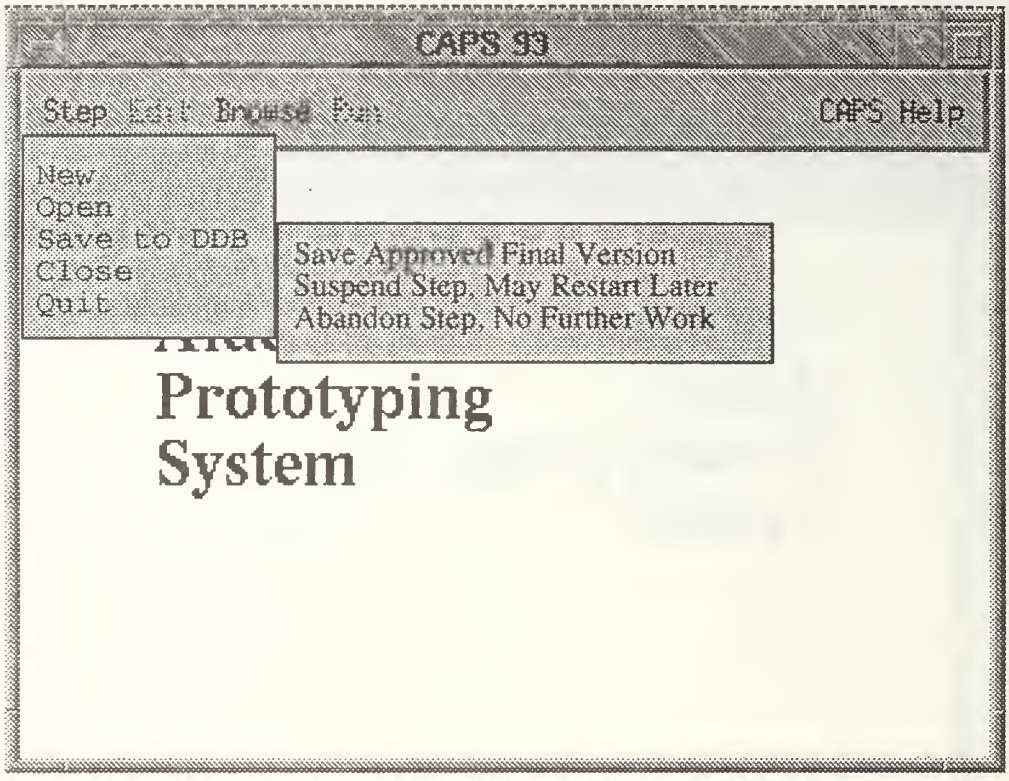

Figure 15. Save Step Menu

There are three ways to save steps. The first option, Save Approved Final Version, is used for finished work that has passed through the appropriate management review process. You are *committing* your work, which makes it available for you and others to create more versions or variations from, but not to modify. The second option is Suspend Step, which also saves your work to the DDB, but in a form allowing it to be retrieved again for further modification. This might be used in a situation where work on a particular step has come to a standstill, but may start up again when new resources available, possibly new reusable code modules, further technical guidance on implementation, or further design information from the project management team. The final save option is to Abandon Step, which saves the step in the DDB in an incomplete state, just like Suspend Step, but no further work is possible. The step may not be changed any further.

Any of the above save options results in your working copy of the step being loaded into the Design Data Base. The user interface will also delete the working copy from your work area. If you want to make further changes, you must check it out from the DDB as a new version or variation.

The next possible option is to close the current step. This merely allows you to put down the current step so you can pick up another one. You may reopen it at any time, providing you haven't saved it to the DDB, abandoned, or suspended it.

Once you have chosen a step to work on, there are now a number of additional top line menu options available, as shown in Figure 9. You may now edit the part of the prototype to be updated by the step or build/run the prototype.

1. Editing

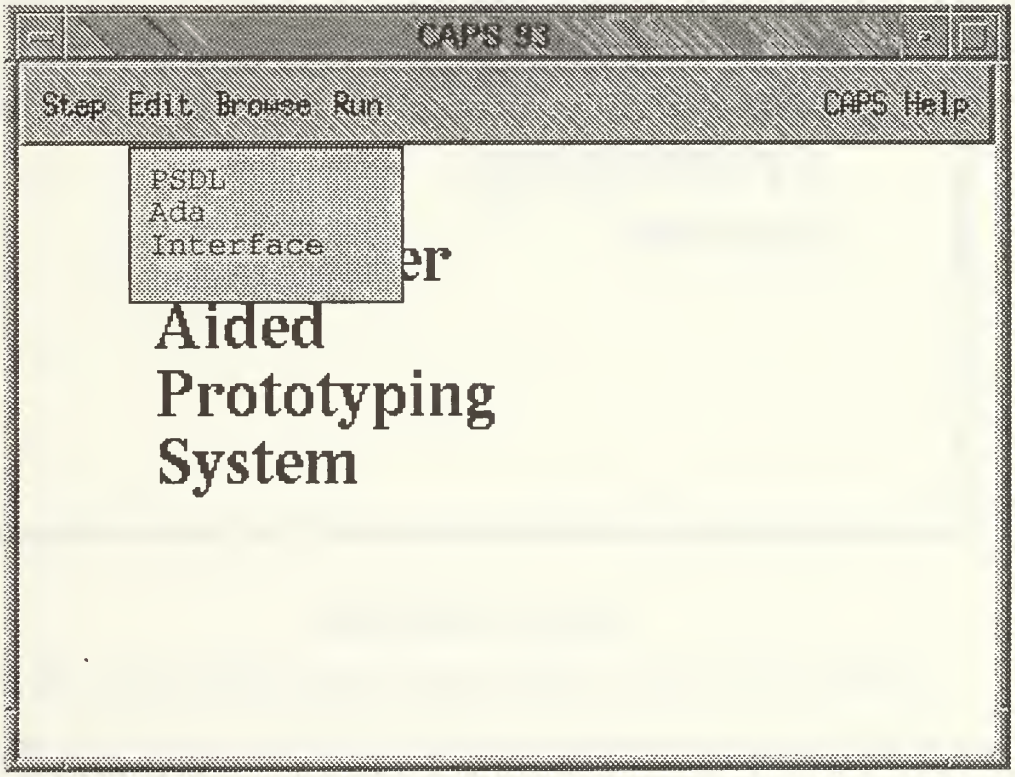

Figure 16. Edit Menu

There are three options available from the Edit menu. The first is to edit the step's Prototype Specification Description Language (PSDL)representation using the PSDL Editor. The second is to edit the Ada code for the atomic operators using a text editor. The third option is to build a user interface for the prototype using the user interface generator.

The use of the PSDL Editor is covered in the next section. The use of the other editors is beyond the scope of this manual.

2. Running a Prototype

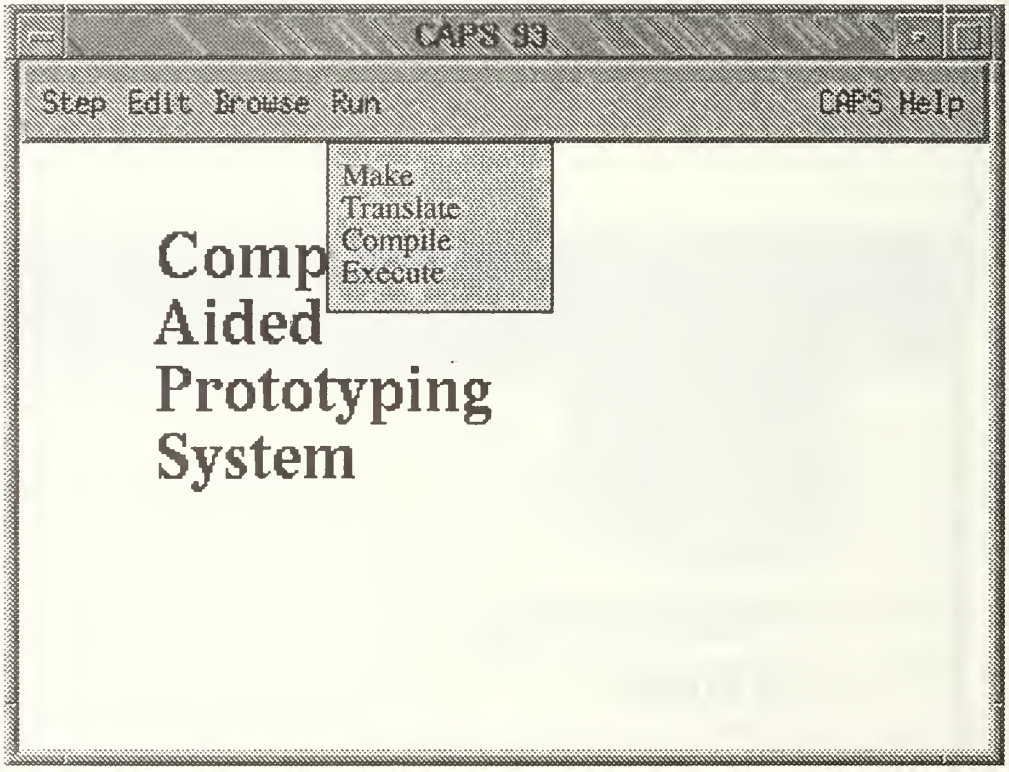

Figure 17. Run Menu

The Run menu has four available options. Make and Run translates the PSDL into Ada, compiles the Ada, and executes the prototype. The other options break this function into separate steps. The Translate option invokes the translator to translate the PSDL into

Ada source code. The Compile option invokes the Ada compiler to compile the available Ada code. The Execute option opens a Unix command shell and executes the prototype.

F. WORKING WITH THE PSDL EDITOR

The PSDL Editor is actually two tools in one. It is composed of a syntax-directed text editor as well as graph editor. This reflects the nature of PSDL, which is a text language describing graphical components. The current version of the syntax-directed text editor was built by Dr. Fernando Naveda using the Cornell Synthesizer-Generator, a toolkit created at Cornell University. Documenting the use of the syntax-directed editor is beyond the scope of this manual. Further information on the Cornell Synthesizer may be found in [REPS89].

1. General Concepts of the PSDL Editor.

The PSDL Editor allows PSDL to be viewed in two ways: text view and graph view. This allows the user to work in whatever view seems the most logical at the time. One of the best features of the PSDL Editor is that the two views are tightly linked together to reduce consistency problems.

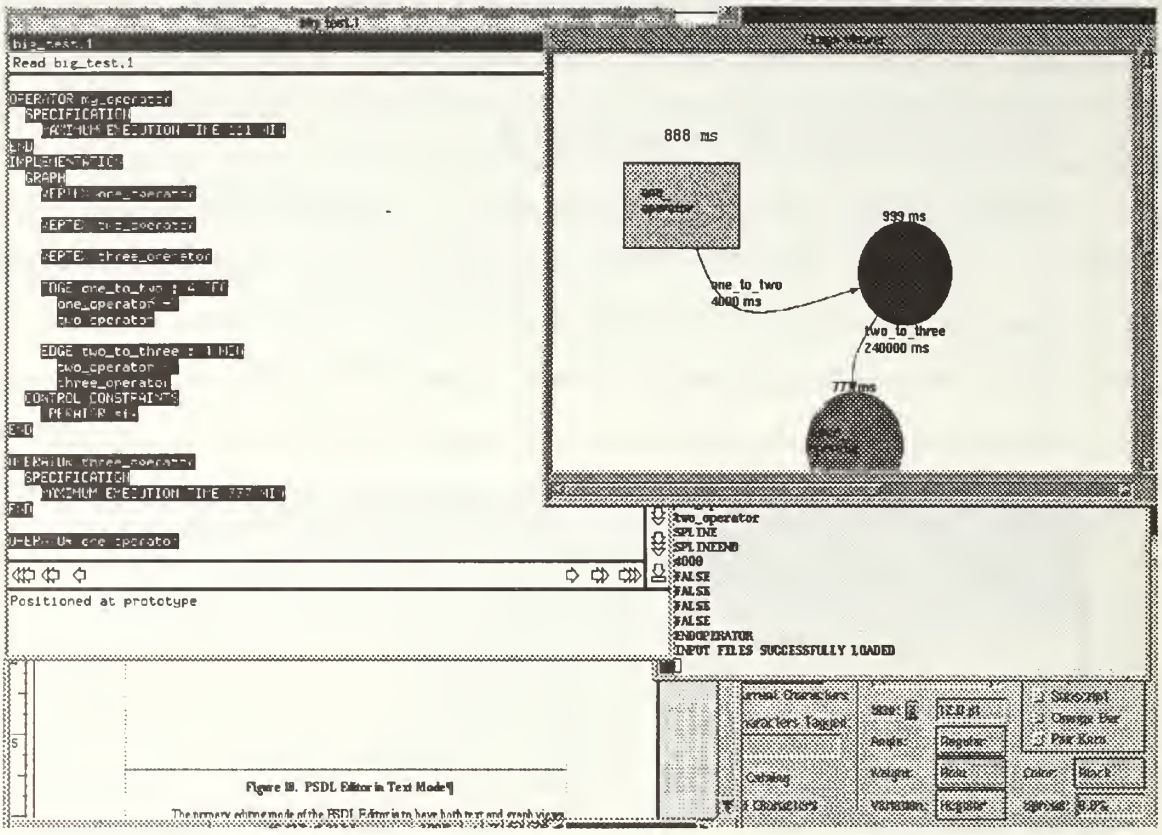

Figure 18. PSDL Editor in Text Mode

The primary editing mode of the PSDL Editor is to have both text and graph views side by side. As changes are made to the text they are reflected in the graph viewer. The graph displayed in the graph viewer window is continually updated to reflect the area of text being edited.

At some point in time the user will undoubtedly want to edit the visual features of the graph, to include the positioning of the objects, color, font, etc.. The secondary editing mode of the PSDL Editor is a full-screen graph editor. This mode allows new graph objects such as operators, terminators, and streams to be added or deleted. It allows these objects to be moved and resized. It allows specification of names and constraints for the objects, as well as their color and font. It also supports the printing of the resulting graph.

2. Interaction Between the Syntax-Directed Editor and the Graph Editor

When the syntax-directed editor is started, it also starts a separate process which runs the graph editor in viewer mode. The graph viewer displays the graph associated with the block of text highlighted in the syntax-directed editor. As the text is changed, the graph automatically updates to reflect the changes.

Eventually the user may want to edit the graph directly. At this point the syntax-directed editor passes the graphic attributes to a second copy of the graph editor operating in full-screen editing mode.

Whereas the graph viewer was run in the background, allowing both tools to run at the same time, the graph editor is run in the foreground. This eliminates possible problems caused by trying to edit the same prototype in both windows. This also has the unpleasant side effect of preventing the syntax-directed editor from updating its window when resized or when portions are covered by other windows and then uncovered. This problem will hopefully be addressed in future versions of the syntax-directed editor.

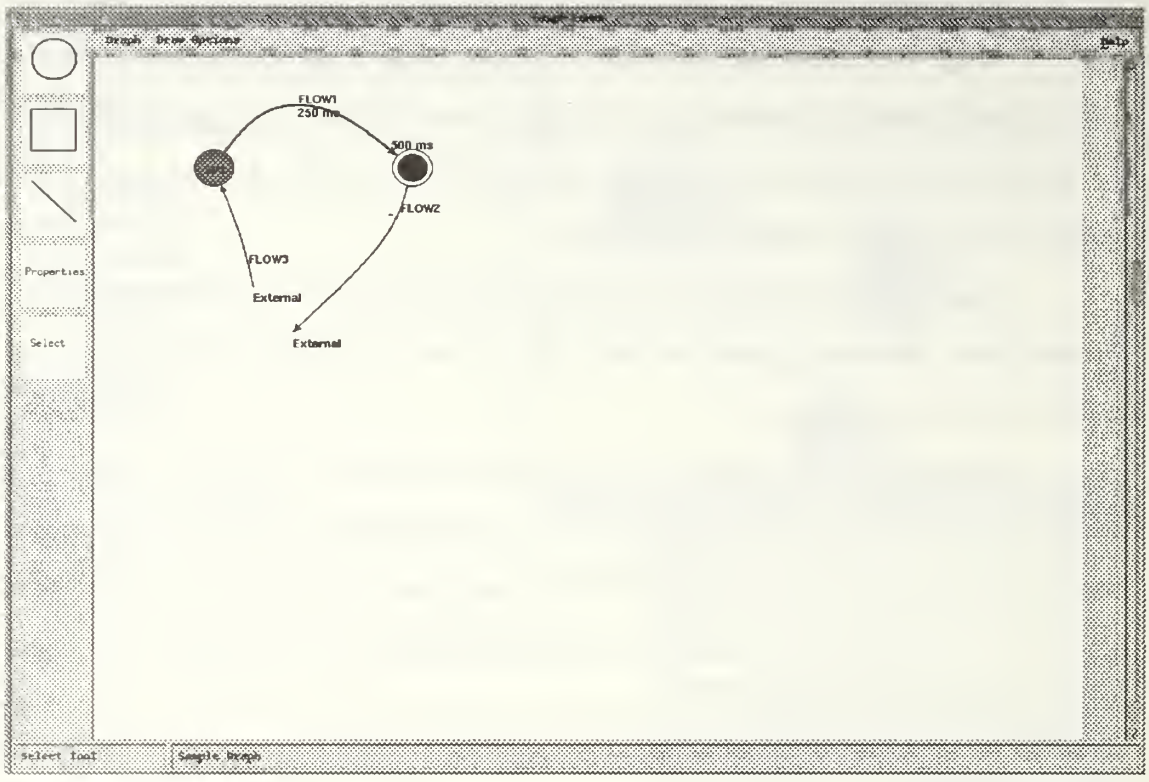

Figure 19. PSDL Editor in Graph Mode

The user is now free to update the graph as desired. When finished, the graph editor first writes out the attributes of the graph and then stops executing. The interface routines within the syntax-directed editor read the attributes and then update the graph information stored in the syntax-directed editor. These changes are then propagated throughout the PSDL description of the prototype.

3. Operation of the Graph Editor

a. General

The graph editor is a full-featured drawing editor created specifically to edit the graphical attributes of PSDL prototypes. It allows ordinary operators and terminators to be placed, moved, resized, and deleted. They may be displayed in a variety of colors, with each operator or terminator assigned a different color, if desired. It allows curved streams to be placed, moved, and deleted. It allows names and constraints to be assigned to

operators, terminators, and streams. These text attributes are assigned a default location, but may then be moved anywhere on the graph. The font used to display text may be changed as desired, with a different font assigned to each text string, if desired. The graph may be printed, and when finished, control returns to the syntax-directed editor, which generates the appropriate PSDL text to describe the editing changes.

b. Mouse Features

Mice in common use today have a variable number of buttons, from one to three. Although the optical mouse in common use with Sun and compatible workstations has three buttons, this may not necessarily always be so.

Applications in common use today are notoriously non-standard in how the buttons on the mouse are to be used. In the interest of simplicity, the graph editor uses only the left button.

c. Specifying defaults

The user may specify a default font and color for the editing session. Do this by selecting Drawing Options from the graph editor top line menu with no graph objects selected. Pull down the appropriate menu, either Colors or Fonts, and select the desired default.

d. Working with Operators and Terminators

Internally, operators and terminators are the same type of object with a different visual appearance. Accordingly, they will be discussed together. It is reasonably tedious to continually say "operators and terminators" or to use "operators/terminators" to refer to the objects under discussion. For simplicity, the use of the word *operators* in this section refers to both operators and terminators.

Operators are placed on the graph by first selecting the Operator Tool. Referring to Figure 19, the Operator Tool is the button with the circle. The Terminator Tool is the button with the rectangle. Either tool is selected by clicking on the button with the mouse. Then, place the operator by clicking in the desired spot on the graph. The Operator

Tool is engaged until another tool is selected, so an operator will be placed every time the mouse is clicked.

Existing operators are modified by first engaging the Select Tool. This is done by clicking on the button marked Select. Then, click somewhere in the operator. Text strings often overlay operators, and have first priority for being selected. To modify the operator vice one of its associated text strings, click on a part of the operator not covered by the text string.

The tool provides visual feedback for this operation by surrounding the selected operator with small square boxes known as *handles*. If one of the operator's text strings is selected, the text string will be surrounded by handles also. To modify only the operator, make sure the text strings don't have handles around them.

To move the operator and its text strings, select the operator and hold the mouse button down. Drag the object to its new location, and release the mouse button. The object will now be redrawn in its new location, with the text strings in the same relative location, and still connected to the same streams, which have moved their attachment points with the operator.

To resize the operator, click on one of its handles and hold the button down. Move the mouse until the operator is the desired size, and then release the button.

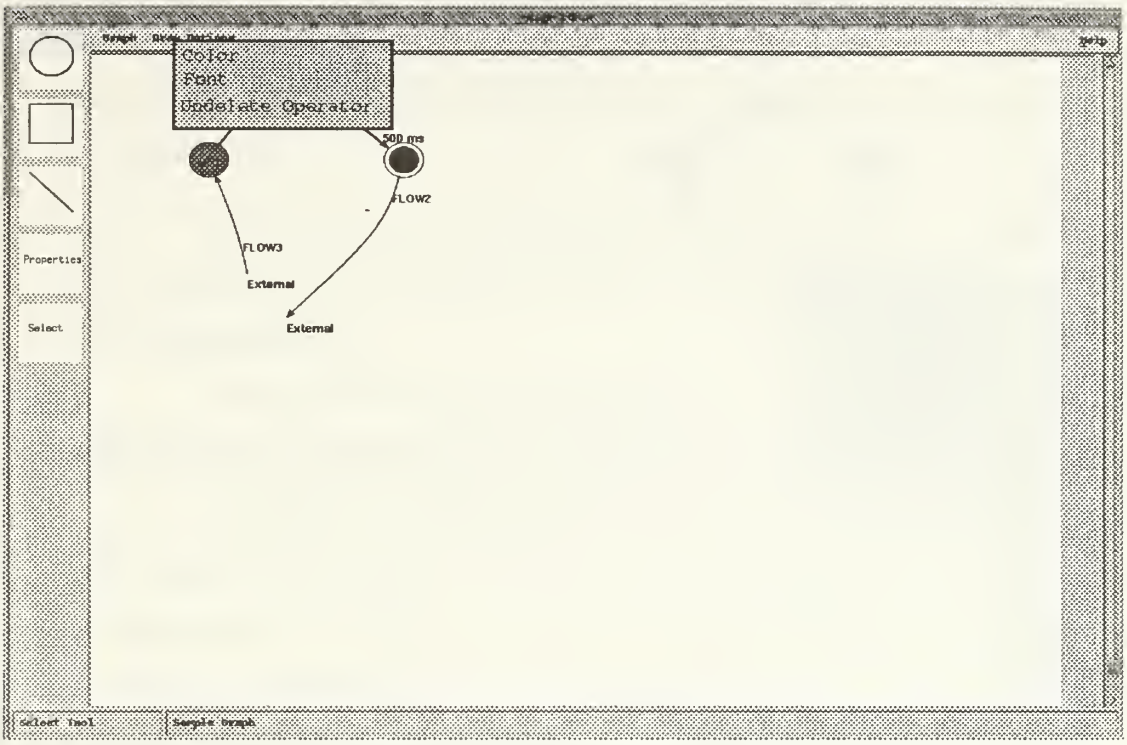

Figure 20. Draw Options Menu

To change the operator's color, select the operator as previously discussed. Go to the menu bar of the graph editor and select Colors from the Draw Options menu, as shown in Figure 20. Click twice in rapid succession on the desired color, which changes the operators color.

To delete an operator, select it and press the Delete or Backspace key on the keyboard. This will cause the operator to disappear along with any attached streams.

To undelete an operator, select the Undelete Operator option from the menu and then select the name of the operator to undelete from the list presented. The operator will reappear, along with any undeleted streams attached to undeleted operators, or any external streams attached to the operator.

To unselect an operator and its text strings, ensure the mouse pointer is not over another object, and click.

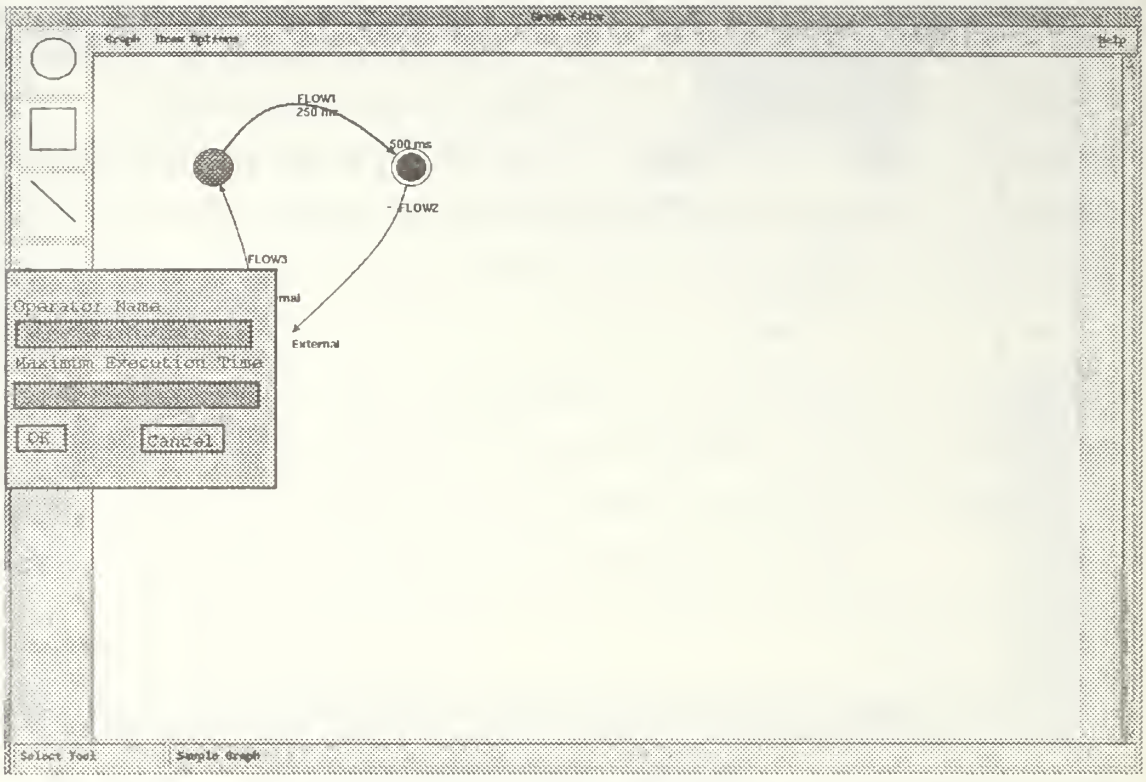

Figure 21. Properties Dialog Box

To change the name of an operator, or to add, change, or delete its Maximum Execution Time (MET), select an operator and engage the Properties tool. This is done by clicking on the button marked Properties. A dialog box pops up allowing the new name and MET to be specified. After entering the name value, press the tab key or use the mouse to move the cursor to the MET field. When the desired values are entered, press the Return key or click on the OK button to apply them to the graph. If a field is left blank, the editor assumes that the existing property should be erased. To exit the dialog without making changes, click on the Cancel button on the dialog box.

e. Working with Streams

To create a new stream, click on the button with the line, as shown in Figure 19. Generally, streams are laid out by specifying the beginning and ending operators as well as the control points used to draw the stream. Different types of streams are specified in

different ways. For a stream connecting two operators on the graph, click first in the operator where the stream begins. Click once at the location of each control point. Finally, click in the operator where the stream ends. The stream will now draw itself, including arrowhead.

For a stream whose beginning point is off the graph, i.e. an external stream, click on the starting point of the stream. Select the remaining control points, and then click in the ending operator.

For a stream whose ending point is an external, click on the beginning operator, select the intermediate control points, and double-click on the endpoint.

Streams are moved in the same general way as operators. First, select the Select Tool. Click anywhere on the stream to select it. It provides visual feedback when selected by showing its handles. A handle is displayed over each control point, including endpoints of external streams. To move the control point, place the cursor over the control point. Press and hold the mouse button. Drag the control point to the desired location. Release the button.

To delete a stream, select it and press the delete or backspace key.

To change the name or add/change the latency value, first select the stream. Click on the Properties Tool, which brings up a dialog box similar to Figure 21. Enter the stream name and the latency, using the tab key to move between fields. If no value is entered, the editor assumes the value should be erased. Select the appropriate toggle button to indicate whether or not the stream represents a state variable. To apply the new field data to the graph, either click on the OK button or press the Enter key. To continue editing without changing the stream, click on the Cancel button.

f. · Working with Text Strings

Changing attributes of text strings is reasonably straightforward. As with the other objects, first engage the Select Tool. Select the string by clicking on it with the mouse. Handles will then appear around the string, and the object the string belongs to will display

its own handles. To change the font, pull down the Draw Options menu as shown in Figure 20 and select Fonts. Double click on the desired font, and the text string will update its appearance. To move the string, place the cursor over the string and hold down the mouse button. Drag the string to its new location, and release the mouse button. To modify the text string, select the Properties tool.

g. Graph Options

Other features are available from the Graph Options menu.

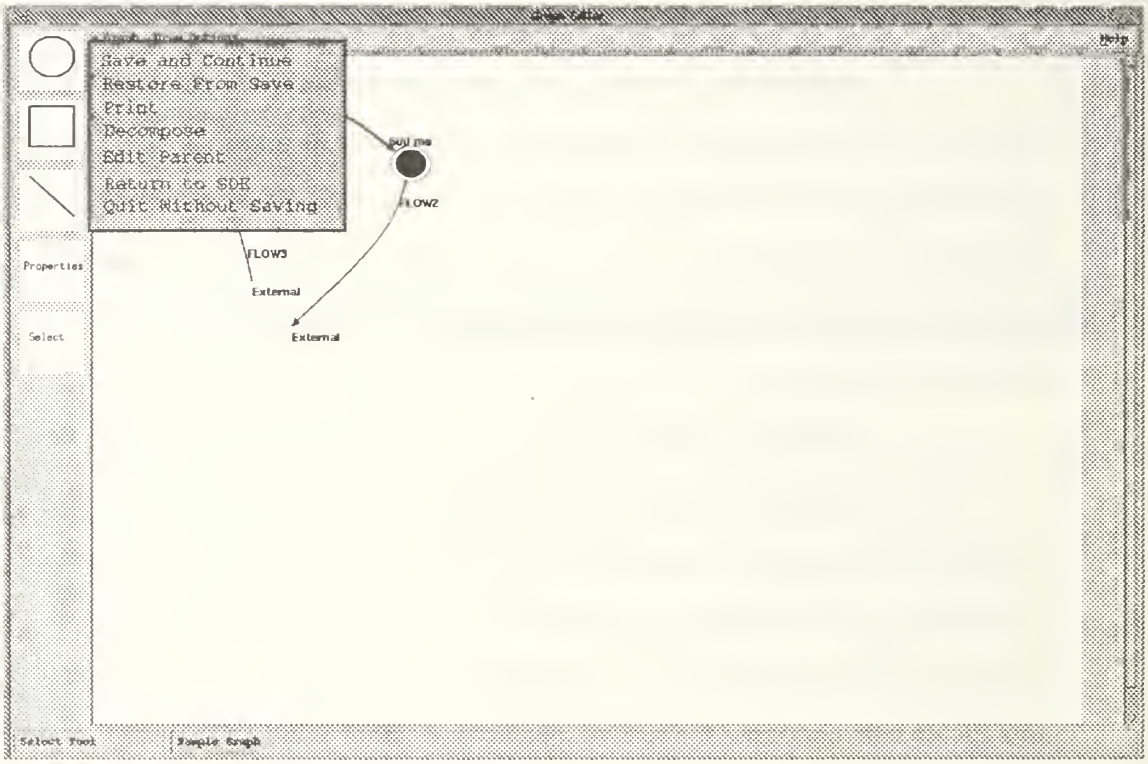

Figure 22. Graph Options Menu

To temporarily save the current graph and keep editing, select the Save and Continue option. To load a previously saved graph, select the Restore from Save option.

The graph is printed by selecting the Print option from the Graph Options menu. The graph is printed using the Unix xwd command at present, which provides a crude but simple printed graph. For best results, expand the window to full-screen, select

Print, and wait for two beeps before continuing. Otherwise, anything else on the desktop will be included in the printed diagram.

To view the parent of the current graph, select View Parent from the Graph Options menu. The current editing session will be terminated, the graph data in the syntax-directed editor will be updated, and a new graph editing session will be invoked by the syntax-directed editor on the parent graph.

To edit the children of an operator on the graph, select the operator using the Select Tool. Select Decompose from the Graph Options menu. The current editing session will be terminated, the graph data in the syntax-directed editor will be updated, and a new graph editing session will be invoked by the syntax-directed editor on the selected operator.

To return to the syntax-directed editor, select Return to SDE from the Graph Options menu.

To return to the syntax-directed editor without saving the changes from this editing session, select Quit Without Saving from the Graph Options menu.

4. Operation of the Syntax-Directed Editor

The syntax-directed editor is still under construction as this manual is written, thus it is not possible to document its operation at this point.

Appendix C

I. PROGRAMMER'S MANUAL

The Programmers Manual has a two-fold purpose. One is to provide more extensive documentation on the inner workings of the menu shell and the graph editor for interested readers. The other is to aid any poor soul tasked with maintaining them.

A. ENVIRONMENT

Each prototype is stored in its own subdirectory under the .caps subdirectory. Within each prototype, each version of the prototype is stored in a subdirectory whose name is the version number. For example, version 1.1 of the fishfarm prototype would be stored in the ~/.caps/fishfarm/1.1 subdirectory. Within each version directory are subdirectories for the prototype files, Ada files, executable files, and temporary files. These subdirectories are named prototype, ada, bin, and temp, respectively. CAPS '93 also automatically makes an Ada library in the ada subdirectory.

B. GENERAL

1. Introduction

The code for this thesis is divided into four major blocks.

- Menu Shell
- Interface to Syntax-Directed Editor
- Graph Editor
- Testbed Syntax-Directed Editor

The menu shell and the graph editor are implemented in C++ using the Motif toolkit for the reasons discussed in Chapter IV of this thesis. The interface routines are implemented in C, because the syntax-directed editor is implemented in C, and it simplifies the linkage process.

The testbed syntax-directed editor provides driver functions for the interface routines to allow them to be tested and debugged without depending on the presence of the actual syntax-directed editor.

2. Inheritance

Although one of the major features of C++ is its use of inheritance, inheritance benefits mainly the programmer, who can add new features to an existing block of code merely by inheriting the old object and adding the new methods and instance variables. To the person stuck with reading the code and trying to maintain it, inheritance effectively takes the functionality of an object and scatters it out over multiple source files, with little or no clue as to where a particular method or instance variable is actually defined.

For this reason the use of inheritance is limited in order to make the code easier to read and maintain. This is the inheritance tree used in the graph editor:

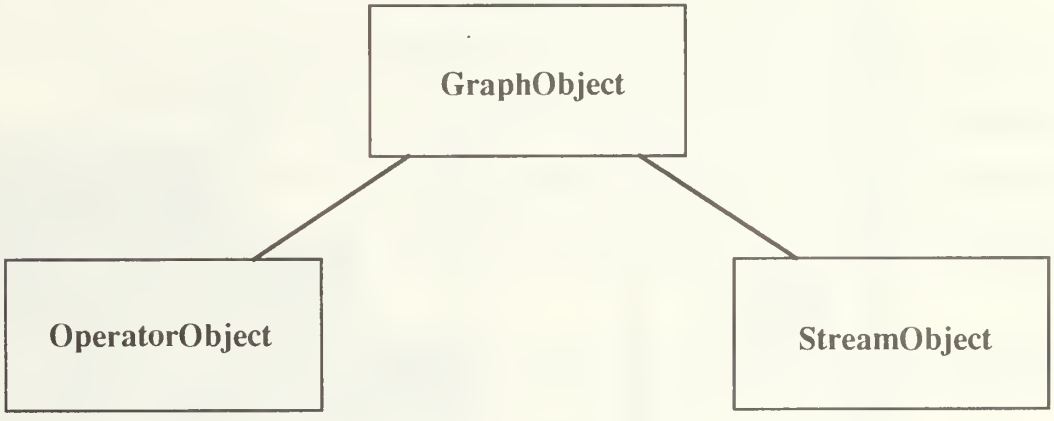

Figure 23. Inheritance Tree for Graph Editor

The only inheritance used in the graph editor is to define a base class, GraphObject, for the two graph object classes, OperatorObject and StreamObject. This makes it possible for the graph editor to send messages to the objects being manipulated without worrying about what type they are. This would also allow a new type of object to

be inherited from GraphObject and added to the editor with minimal modification of existing code.

3. Object-Oriented Methodology

C++ is a superset of C incorporating object-oriented features. A major advantage of this approach is that it allows the programmer a certain amount of flexibility in how to do things. In some object-oriented languages such as Smalltalk and Actor, every data item in the language must be defined as an object and must have its place in the inheritance tree. C++ allows the programmer to tailor the program design to the problem, using object orientation where it makes sense, and avoiding it where it does not.

Even though the majority of the code for this thesis is in C++, functionally the design is a hybrid between object orientation and more conventional design. The reason for this is the structure of the Motif toolkit and X.

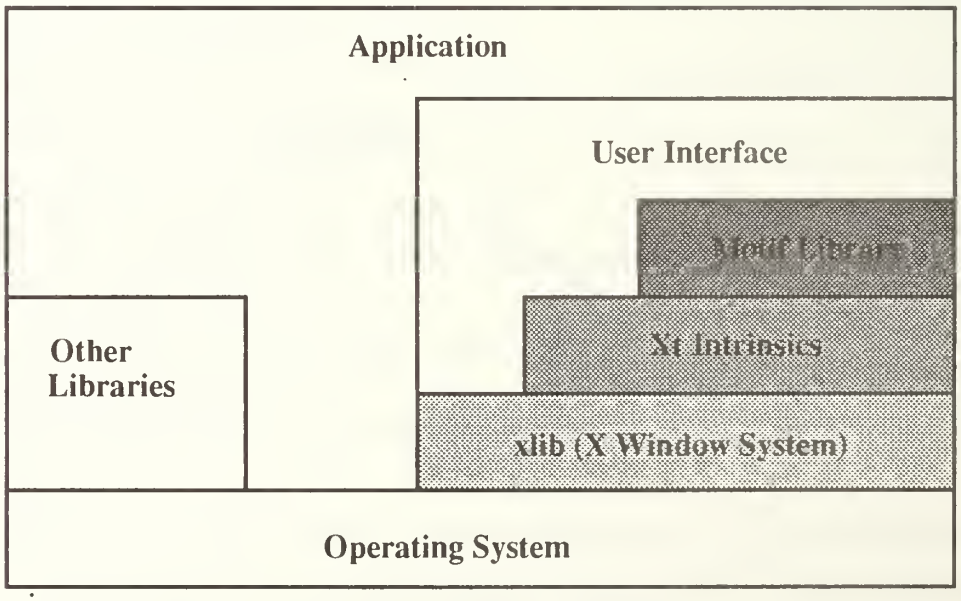

Figure 24. Layers of X

Figure 24 [HELLER91; p. 18] graphically displays the structure of most X programs. The X Window System implements low-level drawing primitives, along with the network interface required to operate under the client-server model. Existing operating

system functions are not duplicated. The Xt Intrinsics level adds support for widgets and callbacks, once again without directly duplicating xlib or operating system functions. The Motif library builds on the previous levels, implementing a number of convenient abstractions for writing user interfaces, also without duplicating the lower-level functions used for drawing graphics primitives, etc..

Motif depends heavily on the use of *callback functions* for the functioning of the user interface. Each interface object, or *widget* in X terms, has a number of predefined hooks where user defined functions are placed. For example, one of the responsibilities of an X program is to keep its window properly drawn when portions are covered by other windows and then exposed. The window manager lets the program know that it must redraw itself by sending it a special message called an *expose event.* In a program written at the X level, the program would spend its time in a semi-infinite loop waiting for these messages to arrive. The program would have to figure what type of event it had received and process it accordingly. This generally results in huge case or switch statements as the program attempts to figure out exactly what type of event it has received.

In Motif, the programmer provides the window manager with the address of a callback function to be called when a particular widget receives a particular event. In our example, the programmer would register a callback function for the expose event that would be called when the expose event was received. This function would probably just redraw the widget.

Callback functions are grouped into predefined types and number of parameters. Generally they have three formal parameters:

- a pointer to the widget where the event took place
- a pointer to user-defined data
- a pointer to a callback structure

The callback structure contains varied information the function might need to process the callback.

This structure limits the way information may be passed. For example, a callback for an expose event would likely redraw the widget. Several data items are necessary to do this:

- a pointer to the data structure describing the physical display being used
- a graphics context describing the attributes to use when drawing
- a pointer to the window being drawn in.

It should also be noted that callback functions are not class methods in the traditional sense, in that they are not functions assigned to a particular data object.

Given all the above, its certainly possible to force Motif into an object-oriented structure. All the required graphics information could be bundled into one object and sent as user-defined data. The callback function could determine which interface object received the event and could call a method of the object involved. The only advantage gained, however, is aesthetics, a certain purity of structure. The disadvantage is that the code becomes more complicated and difficult to read.

In order to simplify the life of follow-on thesis students, I elected to let Motif be Motif. All the data for windows, widgets, and graphics that must be used by multiple callback functions are defined as global variables. Although the major data items used by the program are defined as objects, integers are still integers and strings are still strings. The graph editor is not defined as a separate object.

4. Debug Code

Examination of the source files in Appendix D reveals a considerable amount of debug code that should not be present in a finished product. Which is why it is included. At the time of this writing, many of the parts of CAPS '93 are still under construction, especially the syntax-directed editor. A conscious decision was made to leave this code in as a convenience to follow-on programmers. It can all be turned on by including the -DGE_DEBUG compiler switch, and turned off by not including it. The debug code prints

the values of critical data items during program execution as well as prompts to show what paths the execution took.

There are two functions used to display information about events sent and received: event_list() and id_event(). Both are present in graph_editor.C and id_event() is present in ge_interface.c. Although these functions aren't currently called, they are left in for future use.

C. MODULE DESCRIPTION

As mentioned previously, the implementation part of this thesis can be divided into four major areas: the menu shell, the interface between the syntax-directed editor and the graph editor, the graph editor itself, and the testbed syntax-directed editor. This section provides a description of each area.

1. Menu shell

The function of the menu shell is to tie the various CAPS tools together for the user. It presents the various system options, logically grouped, and allows the user to choose between them.

The source code for the menu shell is contained in the following source files: shell.C, command_table.h, command_table.C, command_node.h, and command_node.C. shell.C implements the main program, caps93. command_table.h is the specification for the CommandTable, an object that associates text tokens with Unix commands to be executed by the appropriate menu options. The CommandTable is initialized from a text file, allowing easy customization of actions the menus perform. command_table.C is the implementation of the CommandTable. command_node.h is the specification for the CommandNode object, which associates one token with a Unix command. command_node.C is the implementation of the CommandNode.

a. Implementation Details.

From a programmer's standpoint the menu shell creates and manages the Motif widgets that provide the menu. Callback functions exist for the menu options, and

proceed accordingly. When the shell is started, it first ensures that a .caps directory is present in the users home directory. If not, it makes one. As the user selects new evolution steps to work on, the shell creates the appropriate directories and allows the user to invoke the desired tools.

The interface between the menu shell and the tools themselves is provided by a CommandTable object. The CommandTable initializes itself by reading a text file that associates certain menu options with Unix commands. These commands may be binary executables, or they may be script files. The intent is to logically isolate the menu code from the execution code in order to simplify maintenance. Ideally, this structure should allow just the appropriate script files to be changed when certain tools are changed or updated, without forcing a recompile of the program. Plus, it provides a certain amount of flexibility to substitute different versions of the various tools for testing and evaluation purposes.

Currently the tool_location.txt file performs two functions. It contains the command strings used to start the graph viewer and graph editor, and it associates string tokens with Unix script files. The tool_location.txt file is kept in the same directory as the syntax-directed editor for convenience. Once CAPS '93 is completed, it should be moved to the /usr/local/caps/bin directory.

All the script files called by the CommandTable object are located in the ~/bin directory, again for convenience during system development. They should eventually be moved to the /usr/local/caps/bin directory also.

2. Interface to Syntax-Directed Editor

The graphic attributes maintained by the syntax-directed editor are defined as global variables, and are maintained as a collection of atomic values and linked lists. The interface module handles the interface between the syntax-directed editor and the graph editor.

When the graph in the graph viewer is updated, or the graph editor is invoked, the global data is written out to a text file. In the case of the graph viewer, the interface module

sends an X event to the viewer notifying it that it must read the data file and update the display. In the case of the graph editor, it reads this data when it starts up.

Communication between the syntax-directed editor and the graph viewer is one-way. The graph viewer merely listens and doesn't talk.

Communication between the syntax-directed editor and the graph editor, however, must be two-way. The graph editor is invoked in the foreground, and when it terminates it writes output data to a separate text file. The interface routines read this file and update the global data structure.

The file expected as input to the graph editor is gedatatransfile.txt, and it has the following format:

- the text string "GE93DATAFILE"
- the current operator name
- the operator name font, x, and y values
- the text string "OPERATORS"
- data for each operator
- the text string "STREAMS"
- the data for each stream, including spline control points between "SPLINE" and "SPLINEEND" strings
- "ENDDATA" to signal the end of data

The output data from the graph editor is stored in the gedatatransfile2.txt file, and follows the above format. After the "ENDDATA" string is the return code to be interpreted by the interface routines linked into the syntax-directed editor. The return code is determined by the menu option in the graph editor used to end the editing session. The return code tells the interface routines whether the user meant to decompose to an operator in the graph, view the parent operator of the current operator, ignore any changes made, or update and continue working at the same level. This information is communicated to the syntax-directed editor by setting two global variables. The first indicates whether or not a level change is desired. If a downward level change is indicated, the second global variable will be set to the name of the operator to decompose.

As discussed above, the syntax-directed editor and the graph editor communicate using text files. A slightly more efficient and possibly more elegant way would be to pass the attributes in a root window property. The interface is implemented with text files vice properties to ease debugging. It is relatively easy to use a text editor to examine the incoming and outgoing data files to locate problems that would be much harder locate in properties. The use of properties would be an advanced feature that should be investigated when CAPS '93 is closer to completion.

The source code for the interface routines is contained in ge_interface.c

a. Implementation details.

ge_interface.c defines three functions used by the syntax-directed editor. These are declared as external functions in the syntax-directed editor, and the script used by the Cornell Synthesizer-Generator is modified to link in this additional object module.

refresh() writes the global data to a file and then calls the send_event() function, which sends a message to the graph viewer telling it to update itself. This happens in the following way.

When the graph viewer is first invoked, it saves the identifier for its display window in a special X data area known as a *property*. Properties are included in X to simplify inter-process communication. Each window has a property area, simply a block of memory set aside for processes to use to share data. In X, when you cut and paste between programs, the data involved is saved in a property to enable the transfer. A property is essentially a collection of named mailboxes definable by the user. One process stores data in a property with an assigned name, and another process may then request the data stored under that name.

Although all windows have properties, its often simpler to use the property area for the *root window*, which is the parent for all the windows in the display, and provides the background for the display. The window identifier for the root window is available to all child windows using the macro call.RootWindowOfScreen().

The interface routine reads the graph viewer's window identifier and then saves a token describing the action the graph viewer is to perform, again as a property of the root window. It then sends a special event known as a *ClientMessage* event to the window identifier extracted from the property area. When the window manager receives the ClientMessage event for the graph viewer, it invokes the callback function registered by the graph viewer. This function reads the property data to determine what actions it should take and acts accordingly.

The edit() function writes the global data to disk and then invokes the graph editor using the Unix system() command. The graph editor is executed in the foreground to prevent the syntax-directed editor from modifying the same data the graph editor is using. The graph editor terminates with a return code that tells the interface routine what to do next. The return code indicates whether to (a) go up or down the structure of the current diagram, (b) abandon any changes made in the editor, or (c) whether there were irreconcilable errors. Based on the return code, the interface reads the output data file from the graph editor and updates the global data structure used by the syntax-directed editor.

The kill_viewer() function is used to shut down the graph viewer. When the syntax-directed editor is ready to stop running, it calls the kill_viewer() function, which sends a ClientMessage event to the graph viewer in the same manner as in the refresh() function. When the graph viewer receives the event it retrieves the token from the property area, which tells it to terminate. At this point it exits.

3. Graph Editor

The graph editor is a full-featured graphic editor/viewer that runs in two modes: view mode, which allows no changes, and edit mode, which allows full editing. Although it has two faces, the graph editor is actually one executable module. A command-line parameter determines whether it will execute as a viewer, allowing no editing, or an editor, supporting full editing functions. For a complete description of the graph editor functions, see Appendix B.

The source code for the Graph Editor is contained in graph_editor.C.

a. Implementation Details.

For ease of reading, the term graph editor includes both editing and viewing modes, unless otherwise specified.

The graph editor is composed of several objects used in the following hierarchy. Note that this only shows what modules use other modules. It implies nothing about inheritance.

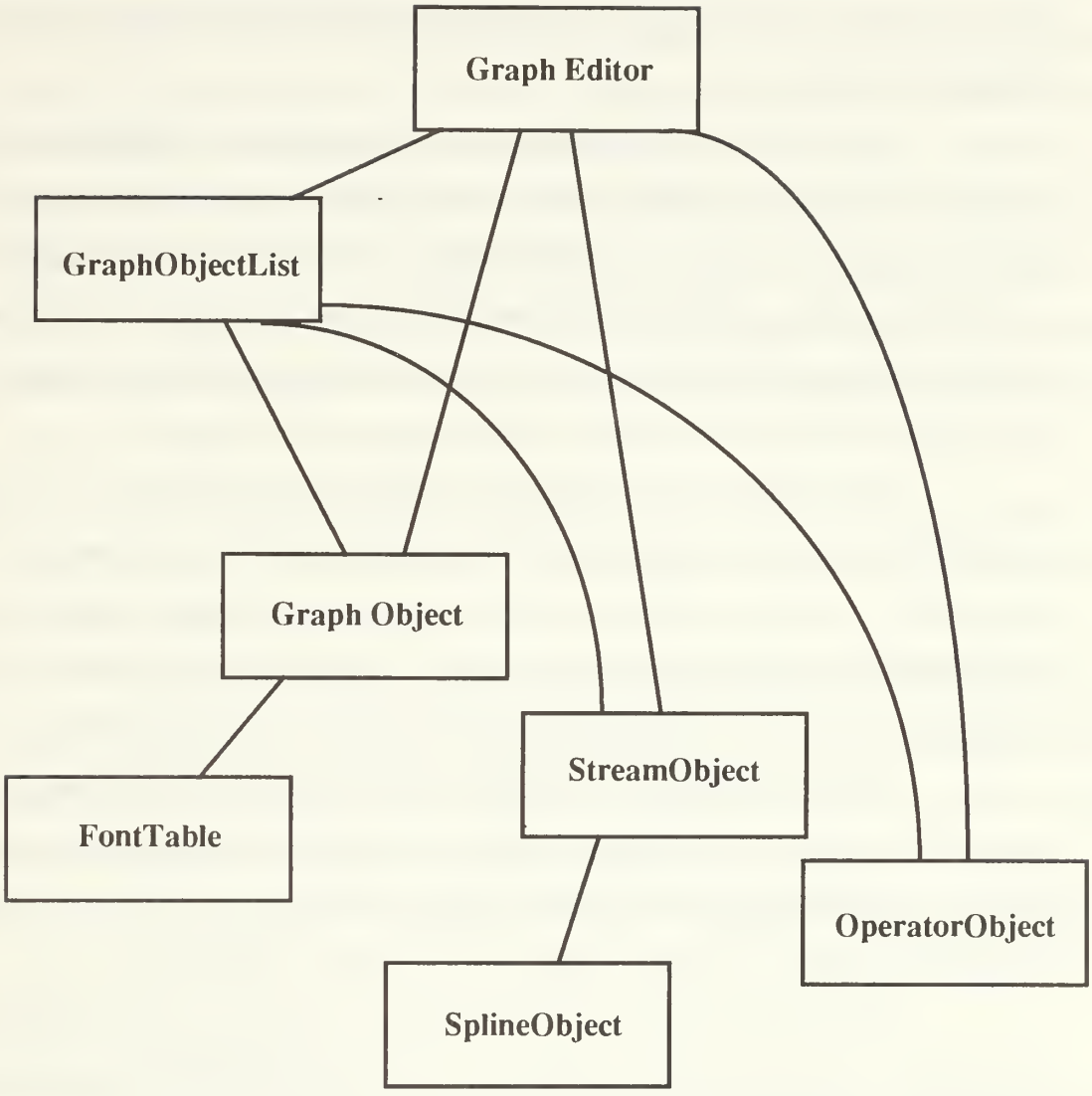

Figure 25. Graph Editor Structure Chart

The graph editor contains one major object, the GraphObjectList, which is a collection of the data objects displayed on the graph. The data objects contain all the specific attributes necessary to display themselves, and they are manipulated by calling methods discussed below. The data objects draw themselves two places: a DrawingArea widget, and on a pixmap. The DrawingArea widget is the drawing canvas actually

displayed, while the pixmap is used to save the appearance of the drawing. When the drawing window must be redrawn, the data in the pixmap is merely copied back onto the DrawingArea widget.

Most of the drawing functions are accomplished or initiated in the draw() function. This function is called any time an event happens in the drawing area. This is enabled by registering draw() as a callback function in the keyboard mapping table. draw() is called with a string parameter indicating what type of event occurred. draw() handles presses and releases of the left mouse button, and movement of the mouse while holding the left button down. It also responds to the backspace and delete keys.

The draw() function determines what event has taken place and responds accordingly. For example, if the Operator Tool is engaged and a mouse button is pressed in the drawing area, the draw() function creates a new operator, adds it to the GraphObjectList, and instructs it to display itself. The draw() function handles everything that affects the appearance of the graph, with the exception of drawing streams.

Due to performance considerations streams are drawn as follows. When the stream tool is engaged and a mouse button is pressed in the drawing area, a separate function is called that draws all the handles and interim appearance cues until the stream is terminated in some way. At this point the stream is drawn and it returns.

If the draw() function were used to draw streams, it would be called every time the mouse moved one pixel in any direction. It would have to erase the previous guide line and draw a new one to provide visual feedback on the shape of the line. In practice this proved to be too slow. The drawing of the guide lines lagged too far behind the movement of the mouse pointer to be acceptable.

Now, when the draw_stream() function is called, it turns off events not connected with stream drawing and manually monitors the event loop, bypassing the overhead of the window manager.

4. GraphObjectList

The GraphObjectList is a linked list of GraphObjects, which will be discussed in detail below. Its main function is to act as a manager for the individual GraphObjects and to encapsulate their functioning from the Graph Editor. For example, when the Graph Editor wants to draw all the objects it calls the draw() method of the GraphObjectList, which in turn calls the draw() method for each GraphObject. When the Graph Editor wants to build the GraphObjectList from disk, it calls the build() method of the GraphObjectList, which repetitively creates new GraphObjects and instructs them to build themselves from the input stream.

The GraphObjectList also provides methods to identify GraphObjects with certain features. For example, hit(x, y) returns a pointer to a GraphObject whose image contains the given coordinates.

Although the GraphObjectList has methods updating graphics characteristics such as fonts, the main function of these methods is to pass this information onto the GraphObject class, which also stores the information. The only thing that the GraphObjectList actually draws is the title of the graph.

The source code for GraphObjectList is contained in graph_object_list.h and graph_object_list.C.

5. GraphObject

The GraphObject class is a base class for OperatorObjects and StreamObjects. Using inheritance in this way allows the GraphObjectList and the Graph Editor to handle GraphObjects without worrying about what kind they are. Plus, it allows for extensibility in the future. A new type of graphical object could be defined, perhaps a TypeObject, and it could be inherited from GraphObject. As long as all of the existing methods for a GraphObject could be defined for a TypeObject, a minimal amount of change to existing code would be required. Any function expecting a GraphObject actual parameter would already accept a TypeObject.

Most of the methods for a GraphObject are virtual, which means that a derived class may substitute methods with the same name and formal parameters. The correct method is selected at run time. An example is the draw() method. In both the GraphObjectList and the Graph Editor, the draw() method for an instance of GraphObject is called, and depending on whether the object is an OperatorObject or a StreamObject, the correct method is called. Suppose, as in the previous paragraph, a TypeObject were to be subsequently defined, which also defined a draw() method. The Graph Editor and GraphObjectList modules would execute using the TypeObjects draw() method, without the need to be recompiled.

The source code for GraphObject is contained in graph_object.h and graph_object.C.

a. Implementation Details

Although much of the GraphObject's implementation serves as a base class for its child classes, it has some features of its own. All the graphic information necessary for its child classes to draw themselves is contained in static class variables, and is maintained by static class methods.

Although *static* is often used in C to make local variables persistent and for scoping rules, it has another meaning in C++. When a variable is defined in a C++ class, that variable normally has one occurrence per instance of the class. For example, each instance of the OperatorObject has its own set of x and y coordinates. When a variable is declared *static*, however, there is only one instance for the entire class, which all instances of the class have access to. As part of the Graph Editor, there is only one DrawingArea and pixmap on which the objects need to draw themselves, and only one graphics context for each type of drawing operation. Instead of repeating this information for each object, one copy is maintained centrally for the GraphObject class and the instances of its descendants.

To maintain the value of these static variables, static methods are required. In the same way that static variables apply to an entire class, static methods are also used by the entire class.

When a non-static C++ method is called, there is actually a hidden formal parameter, which is a pointer to the instance of the class on which the method is to operate. Static methods don't have this hidden parameter, so they will not act on any particular instance of a class. Static methods can be used in this way to update static class variables, or they can be used for operations that don't alter any of the variables of a particular class instance. An example of this might be a routine that checked the system time or that calculated a trigonometric function. In this case, static methods are used to maintain the static graphics information.

It is worth noting that Motif callback functions must be defined as static also. C++ performs strict type checking at compile time, and the hidden parameter of a non-static callback function would prevent it from matching the callback function types used in Motif.

6. OperatorObject

The OperatorObject is derived from the GraphObject and implements the required functionality for the graphic representation of a PSDL operator. It includes x-y coordinates, color, size, text string location and font, and everything else needed to build and maintain itself. It also has attributes indicating whether it is a standard operator or a terminator, whether it is composite, or whether it has been deleted or not.

All the intelligence for the object is contained in the object to provide both abstraction and encapsulation. OperatorObjects know how to build themselves from disk, display themselves, and write themselves to disk. To determine whether an object was selected or not, the OperatorObject provides a method to compare the x-y "hit" coordinates with the location and dimensions of the object, and additionally the text strings associated with it. If selected, the OperatorObject draws its own handles. To simplify locating the beginning and ending points of streams, the OperatorObject provides an intercept point on

the circumference of the object, given a reference point the stream is drawn from.This simplifies maintenance of the class, as well as simplifying the addition of new descendants of GraphObject.

It should be noted that OperatorObjects are never actually deleted. There is a flag in each object indicating whether it has been marked for deletion or not. When the deletion flag is marked, the object is not displayed, but it is not removed from the GraphObjectList. This allows the operator to be "undeleted" at a later date. Deleted objects may be physically removed from the data structure by the syntax-directed editor, although not currently implemented that way.

The source code for OperatorObject is contained in operator_object.h and operator_object.C.

7. StreamObject

The StreamObject is derived from the GraphObject and implements the required functionality to graphically represent a PSDL stream. In the same way as the OperatorObject, it contains all the information necessary to determine its physical appearance and whether its components have been selected or not.

a. Implementation

In addition to the attributes for deletion, etc., the StreamObject also maintains pointers to the OperatorObjects connected to either end. In this way, streams automatically connect to the operators they attach to, even when the operators move and resize. In the case of an external stream, the appropriate identifier is set to 0, and the pointer is set to NULL.

This is not a particularly elegant scheme, because it violates the encapsulation of the OperatorObject. The use of pointers means that extra care must be taken to ensure that the operators on either end of the stream are always there or are not deleted. Otherwise a segmentation fault results from trying to access the operator.

The connection is implemented this way for simplicity. It makes it rather easy to call the OperatorObject's methods directly to find out its location, intercepts, etc..

The appearance of the actual stream, less its text strings, is maintained by a SplineObject, which also draws the arrowhead.

8. SplineObject

As streams in CAPS '93 are implemented as curved lines, a SplineObject is used to maintain and display the physical appearance of the stream. Each stream is specified as a collection of control points between operators. Using b-splines, the curves are drawn to intersect the operators at their circumference. The SplineObject also draws the arrowhead associated with the stream.

a. Implementation

In order to make the b-spline begin and end at the right point, extra control points known as *shadow points* are defined. Normally, b-splines do not actually end on their first and last control points. Shadow points are defined as invisible control points to make the b-splines actually start and stop where the stream should intersect the operator. Shadow points are not displayed on the graph, and they are not stored in the syntax-directed editor. They are recalculated each time any of the control points in the stream or any connected operators are moved.

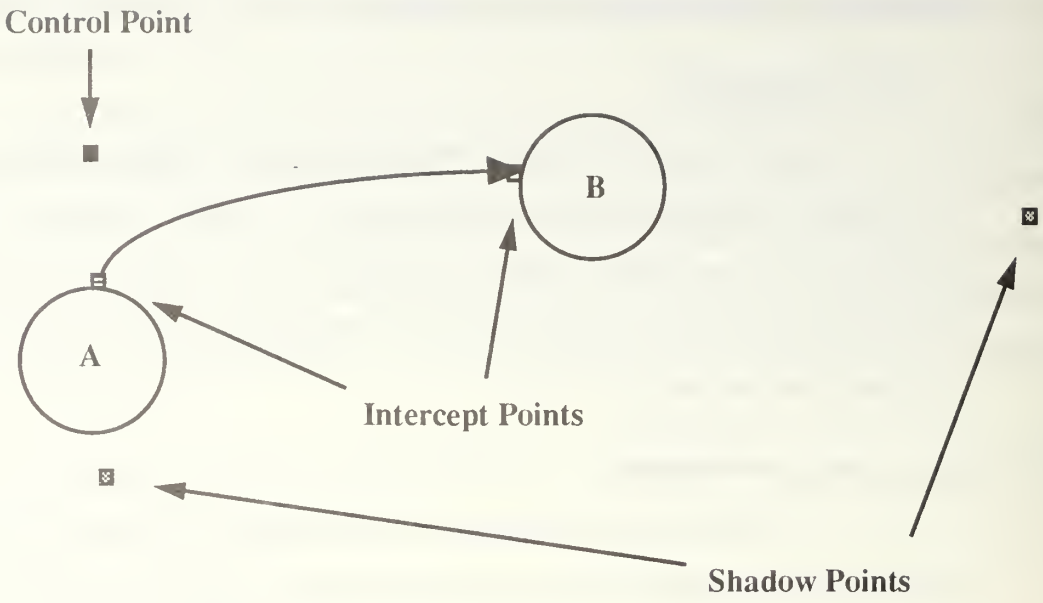

Figure 26. Shadow Points

In the case of a stream with an external beginning or end, the last control point on the respective end is considered to be the intercept point, and the shadow points are calculated accordingly.

If a stream is defined with no control points, which usually happens when a stream is created in the syntax-directed editor, a control point is added at the midpoint of the line connecting the centers of the operators.

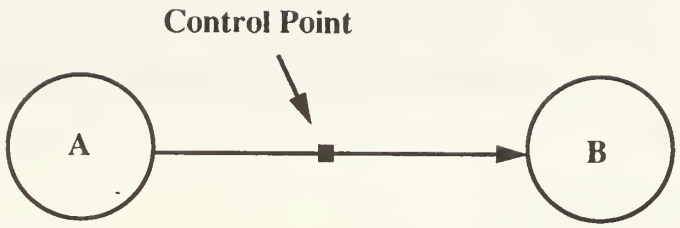

Figure 27. Constructed Center Point

Once the control points are established, the real math begins.

The coordinates of each pixel displayed in a b-spline is calculated according to the following formula:

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \bullet \frac{1}{6} \bullet \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

where $0 \leq t \leq 1$ and P_n is the control point in $\begin{bmatrix} x & y \end{bmatrix}$ format.

Each value of t is mapped to a pixel displayed on the graph, based on the four control points used in the equation. The entire curve is generated by using successive sets of four control points. For example, if there are six control points in the curve, the first subcurve is generated using points 1, 2, 3, 4. The second uses 2, 3, 4, 5, and the final subcurve uses 3, 4, 5, 6. As b-splines are continuous in the first and second derivatives at the points where the subcurves meet, the entire curve is continuous [ZYDA90].

One challenge here is to use successive values of t that are close enough to make a continuous graph, but not so close that an excessive number of calculations must be made. The SplineObject determines the number of pixels that must be drawn by comparing the distances between all the control points, and using the largest difference in x or y coordinates between any two of the control points. Δt is the reciprocal of this number.

To determine whether a b-spline was "hit" or not, the SplineObject uses a rather simplistic method. It successively calculates points in the b-spline and checks to see whether the point calculated is within a specified distance of the mouse click location. It continues until a point is found within the specified distance or all the points have been calculated.

9. FontTable

The FontTable serves as a lookup table for the fonts implemented in the Graph Editor. Six fonts have been included for demonstration purposes, but any number of fonts can be included. To add more fonts, determine the text identifiers of the fonts desired and add them to the FontTable::init() method, remembering to adjust the MAXFONTS constant in resources.h, and to add new integer identifiers for each font in the same file.

Appendix D

SOURCE CODE

```
/* ************************************************************

Name:           command_node.h
Author:         Capt Robert M. Dixon
Program:        caps93
Date Modified: 19 Sep 92
Remarks:   Specification for the CommandNode class. The
          CommandNode is used by the CommandTable to associate
          string tokens with Unix commands.

************************************************************ */
#ifndef COMMAND_NODE_H
#define COMMAND_NODE_H 1

#include <stdio.h>
#include <stdlib.h>
#include "../ge/ge_defs.h"

typedef int STATUS;

class CommandNode {
friend CommandTable;
protected:
  char *option, *command;
public:
  CommandNode() {option = NULL; command = NULL;};
  ~CommandNode() {free(option); free(command);};
  BOOLEAN build(FILE *infile);
};

#endif
```

```
/* ************************************************************

Name:           command_node.C
Author:         Capt Robert M. Dixon
Program:        caps93
Date Modified: 19 Sep 93
Remarks:    Implementation of the CommandNode class.

************************************************************* */
#include <string.h>
#include "command_node.h"

//   Builds a CommandNode from disk.

BOOLEAN CommandNode::build(FILE *infile) {
  char buffer[INPUT_LINE_SIZE + 1];
  int last_char_pos;
  STATUS status = SUCCEEDED;

  fscanf(infile, "%s", buffer);
  last_char_pos = strlen(buffer) - 1;
  if(buffer[last_char_pos] == ':')
    buffer[last_char_pos] = NULL;
  option = strdup(buffer);
  fgets(buffer, INPUT_LINE_SIZE, infile);
  last_char_pos = strlen(buffer) - 1;
  if(buffer[last_char_pos] == '\n')
    buffer[last_char_pos] = NULL;
  command = strdup(buffer);
  if(ferror(infile)) {
    printf("Error building CommandTable node.\n");
    clearerr(infile);
    status = FAILED;
  }
  if(feof(infile))
    status = FAILED;
  return status;
}
```

```
/* ************************************************************

Name:           command_table.h
Author:         Capt Robert M. Dixon
Program:        caps93
Date Modified: 19 Sep 92
Remarks:   Specification for a CommandTable object. A
         CommandTable associates a Unix command with a string
         token. The <execute> method is called with a token as
         a parameter, and it executes the command associated
         with the token.

************************************************************* */
#ifndef COMMAND_TABLE_H
#define COMMAND_TABLE_H 1
#include <string.h>
#include "command_node.h"
#define MAXCOMMANDS 100

typedef int STATUS;

class CommandTable {
protected:
  CommandNode commands[MAXCOMMANDS];
  int last_command_index;
public:
  void init();
  void execute(char *option, char *home_dir,
               char *prototype_name, char *prototype_ver);
};

#endif
```

```
/* ************************************************************

Name:           command_table.C
Author:         Capt Robert M. Dixon
Program:        caps93
Date Modified: 19 Sep 92
Remarks:    Implementation of the CommandTable class. The
          CommandTable associates a string token with a Unix
          command. The execute() method is called with a token,
          causing the associated method to be executed.

************************************************************* */
#include <stdlib.h>
#include <stream.h>
#include "command_table.h"

#define COMMAND_LINE_SIZE 150

//   Initializes the command table from a text file.

void CommandTable::init() {
  FILE *infile;
  STATUS status = SUCCEEDED;
  int index = 0;

  infile = fopen("bin/tool_location.txt", "rt");
  if(infile != NULL) {
    while(status == SUCCEEDED) {
      status = commands[index].build(infile);
      index++;
    }
    last_command_index = index - 1;
    fclose(infile);
  }
  else
    printf("Unable to open tool location file.\n");

}

//   Executes the command associated with the option string.
// The directories are passed to the method so that they can
// be passed to the command being executed.

void CommandTable::execute(char *option, char *home_dir,
                           char *prototype_name,
                           char *prototype_ver) {
  int index = 0;
  char buffer[COMMAND_LINE_SIZE];

  if(last_command_index > 0) {
    while(strcmp(option, commands[index].option) != 0) {
      index++;
      if(index > last_command_index) {
        cout << option << ": command not found." << endl;
```

```
            return;
          }
        }
        sprintf(buffer, "%s %s %s %s", commands[index].command,
                home_dir, prototype_name, prototype_ver);
        system(buffer);
      }
      else
        cout << "Command Table not available." << endl;
}
```

```
/* ************************************************************

Name:           shell.C
Author:         Capt Robert M. Dixon
Program:        caps93
Date Modified: 19 Sep 92
Remarks:          shell.C is the main module for the caps93
          program. It displays the CAPS '93 main menu and
          executes appropriate entries in the command table for
          selected menu options.

************************************************************* */

#include <stdlib.h>
#include <Xm/MainW.h>
#include <Xm/SelectioB.h>
#include <Xm/DrawingA.h>
#include <stream.h>
#include <Xm/Form.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pwd.h>
#include <sysent.h>
#include <errno.h>
#include <dirent.h>
#include <Xm/List.h>
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
#include <Xm/PushBG.h>
#include "strstack.h"
#include "command_table.h"

// Unix system-defined variable to indicate which error occurred.
extern int errno;

typedef int BOOLEAN;

#define TRUE 1
#define FALSE 0
#define MAXDIRECTORIES 100
#define MAXLINESIZE 100

#define NEW_STEP 112
#define NEW_EXISTING 108
#define OPEN 69
#define SAVE_FINAL 87
#define SUSPEND 67
#define ABANDON 84
#define CLOSE 109
#define QUIT 61
#define EDIT_PSDL 70
#define EDIT_ADA 86
```

```
#define EDIT_INTERFACE 68
#define BROWSE_DDB 88
#define BROWSE_SB 111
#define MAKE 113
#define TRANSLATE 65
#define COMPILE 92
#define EXECUTE 63
#define HELP 89
#define ALT 26

Widget main_window, new_button, open_button, save_button,
  close_button, edit_button, run_button;
char *home_dir, *prototype_name, *prototype_ver,
  **directories = NULL;
BOOLEAN dir_set = FALSE, alt_key_selected = FALSE;
CommandTable command_table;
GC white_context, black_context;

// Checks for a ~/.caps directory. If not found, makes one.
// Initializes the command table from the tool_location.txt file.

void init() {
  struct passwd *user_pass;
  int status;
  mode_t fc_mode = 0777;

  user_pass = getpwuid(getuid());
  home_dir = strdup(user_pass->pw_dir);
  chdir(home_dir);
  status = chdir(".caps");
  if((status != 0) && (errno = ENOENT)) {
    if (mkdir(".caps", fc_mode) == -1)
      perror("mkdir()");
  }
  chdir(home_dir);
  prototype_name = strdup("void");
  prototype_ver = strdup("void");
  command_table.init();
}

// Destroys the widget

void widget_killer(Widget widget, XtPointer, XtPointer) {

  XtDestroyWidget(widget);
}

//   Allows appropriate menu choices when a current step
// is not selected.  Allows a step to be selected.

void set_step_state_false() {

  XtSetSensitive(save_button, False);
  XtSetSensitive(close_button, False);
```

```
  XtSetSensitive(edit_button, False);
  XtSetSensitive(run_button, False);
  XtSetSensitive(new_button, True);
  XtSetSensitive(open_button, True);
}

//   Allows appropriate menu choices when a current step is
// selected.

void set_step_state_true() {

  XtSetSensitive(save_button, True);
  XtSetSensitive(close_button, True);
  XtSetSensitive(edit_button, True);
  XtSetSensitive(run_button, True);
  XtSetSensitive(new_button, False);
  XtSetSensitive(open_button, False);
}

//   Gets a new prototype name entered in a dialog box and
// makes a subdirectory for it. Also makes a directory for
// the new step. Executes a script to make required new
// directories in step subdirectories.

static void get_proto_cb(Widget widget, XtPointer,
                         XtPointer cbs) {
  XmSelectionBoxCallbackStruct *temp_ptr;
  char *text;
  mode_t fc_mode = 0777;

  temp_ptr = (XmSelectionBoxCallbackStruct *) cbs;
  XmStringGetLtoR(temp_ptr->value, XmSTRING_DEFAULT_CHARSET,
                  &text);
  XtDestroyWidget(widget);
  prototype_name = strdup(text);
  prototype_ver = strdup("1.1");
  chdir(home_dir);
  chdir(".caps");
  if (mkdir(prototype_name, fc_mode) == -1)
    perror("mkdir()");
  chdir(prototype_name);
  if (mkdir(prototype_ver, fc_mode) == -1)
    perror("mkdir()");
  else {
    chdir(prototype_ver);
    command_table.execute("make_dirs", home_dir, prototype_name,
                          prototype_ver);
  }
  chdir(home_dir);
  set_step_state_true();
  free(text);
}

//   Compiles a list of subdirectories in the given
```

```
// directory, and returns the list to the caller.

char** read_directory(char *directory) {
  char **directories =
    (char **) malloc(MAXDIRECTORIES * sizeof(char *));
  DIR *dirp;
  struct dirent *entry;
  struct stat buf;
  char fname[130];
  char *searchdir;
  strlist list;
  int i = 0, index = 0;

  searchdir = strdup(directory);
  dirp = opendir(searchdir);
  if(dirp != NULL) {
    entry = readdir(dirp);
    while(entry != NULL) {
      if((strcmp(entry->d_name, ".") != 0)
         && (strcmp(entry->d_name, "..") != 0)) {
        sprintf(fname, "%s/%s", searchdir, entry->d_name);
        if(!stat(fname, &buf)) {
          if(buf.st_mode & S_IFDIR) {
    //Adds the entry to the alphabetized list
            list.add(entry->d_name);
          }
        }
      }
      entry = readdir(dirp);
    }
    index = 0;
    while(list.notempty()) {
      directories[index] = list.remove();
      index++;
    }
  }
  closedir(dirp);
  directories[index] = NULL;

#ifdef GE_DEBUG

  for(i = 0; i < index; i++)
    printf("%s\n", directories[i]);

#endif

  free(searchdir);
  return directories;

}

//   Appends two strings together, allocating adequate memory
// for the longer string.
```

```
char *cat_strings(char *str1, char* str2) {
  char *new_string;

  new_string = (char *) malloc((strlen(str1) + strlen(str2) + 1)
                               * sizeof(char));
  strcpy(new_string, str1);
  strcat(new_string, str2);
  return new_string;
}

//   Called when a step name is selected from the Open menu.
// Updates global variables to indicate which step is the current
// one.

static void step_list_cb(Widget widget, XtPointer,
                         XtPointer cb_struct_ptr) {
  XmListCallbackStruct *list_struct_ptr =
    (XmListCallbackStruct *) cb_struct_ptr;
  int dir_selected = list_struct_ptr->item_position, index;
  char *temp_name, *temp_ver;

  XtDestroyWidget(widget);
  if(strcmp(directories[dir_selected - 1], "Cancel") != 0) {
    temp_name = strdup(directories[dir_selected - 1]);
    index = strlen(temp_name) - 1;
    while(temp_name[index] != '.')
      index--;
    index--;
    while(temp_name[index] != '.')
      index--;

// Strips off version/variation number

    temp_name[index] = NULL;
    prototype_name = strdup(temp_name);
    temp_ver = &(temp_name[index + 1]);
    prototype_ver = strdup(temp_ver);

#ifdef GE_DEBUG
    cout << prototype_name << "   " << prototype_ver << endl;
#endif

    set_step_state_true();
    free(temp_name);
  }
}

//   If the Open option is selected, presents a list of steps
// for the user to select from.

void get_open_choice(char *directory) {
  char **temp_directories, **temp_versions,
  search_directory[MAXLINESIZE], temp_string[MAXLINESIZE];
  int index1 = 0, index2 = 0, index3 = 0, i;
```

```
  XmStringTable step_list;
  Widget list_box;

  if(directories != NULL) {
    while(directories[index1] != NULL) {
      free(directories[index1]);
      index1++;
    }
    free((char *) directories);
    index1 = 0;
  }
  directories =
    (char **) malloc(MAXDIRECTORIES * sizeof(char **));
  temp_directories = read_directory(directory);
  while(temp_directories[index2] != NULL) {
    sprintf(search_directory, "%s/%s", directory,
            temp_directories[index2]);
    temp_versions = read_directory(search_directory);
    sprintf(temp_string, "%s.", temp_directories[index2]);
    free(temp_directories[index2]);
    while(temp_versions[index3] != NULL) {
      directories[index1] = cat_strings(temp_string,
                                        temp_versions[index3]);
      free(temp_versions[index3]);
      index1++;
      index3++;
    }
    index3 = 0;
    index2++;
    free((char *) temp_versions);
  }
  free((char *) temp_directories);

// Adds the "Cancel" option to list

  directories[index1] = strdup("Cancel");
  index1++;
  step_list = (XmStringTable)
    XtMalloc(MAXDIRECTORIES * sizeof(XmString *));
  i = 0;
  directories[index1] = NULL;
  for(i = 0; i < index1; i++) {
    step_list[i] = XmStringCreateSimple(directories[i]);
  }
  list_box = XmCreateScrolledList(main_window, "Steps", NULL,
                                  0);
  XtVaSetValues(list_box,
                XmNitems, step_list,
                XmNitemCount, index1,
                XmNvisibleItemCount, 10,
                NULL);
  for(i = 0; i < index1; i++)
    XmStringFree(step_list[i]);
  XtFree((char *) step_list);
```

```
  XtAddCallback(list_box, XmNdefaultActionCallback,
                step_list_cb, NULL);
  XtManageChild(list_box);
}

/*   Handles step menu choices, either from menu or alt-key
   combination.
*/

void handle_step_cb(int item_no) {
  char *open_choice = NULL;

  switch(item_no) {
  case 1:
    get_open_choice(".caps");
//    free(open_choice);
    break;
  case 3:
    set_step_state_false();
    break;
  case 4:
    exit(0);
    break;
  }
}

/*   Callback function for the step menu. */

static void step_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;

  handle_step_cb(item_no);
}

/*   Handles step menu choices, either from the menu or from an
   alt-key combination.
*/

void handle_new_step_cb(int item_no) {
  Widget dialog;
  XmString t;
  Arg args[3];

  switch(item_no) {
  case 0:
    t = XmStringCreateSimple("Enter Prototype Name:");
    XtSetArg(args[0], XmNselectionLabelString, t);
    XtSetArg(args[1], XmNautoUnmanage, False);
    XtSetArg(args[2], XmNuserData, 0);
    dialog = XmCreatePromptDialog(main_window, "New Options",
                                  args, 3);
    XmStringFree(t);
    XtAddCallback(dialog, XmNokCallback, get_proto_cb, NULL);
    XtAddCallback(dialog, XmNcancelCallback, widget_killer,
```

```
                    NULL);
    XtUnmanageChild(XmSelectionBoxGetChild(dialog,
                                           XmDIALOG_HELP_BUTTON));
    XtManageChild(dialog);
    XtPopup(XtParent(dialog), XtGrabNone);
//    XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
    break;
  case 1:
    command_table.execute("new_from_existing", home_dir,
                          prototype_name, prototype_ver);
    set_step_state_true();
    break;
  }
}

/*   Callback function when New Step is selected. */

static void new_step_cb(Widget, XtPointer client_data,
                        XtPointer) {
  int item_no = (int) client_data;

  handle_new_step_cb(item_no);
}

/*   Handles edit menu choices from either the menu or alt-key
   combination.
*/

void handle_edit_cb(int item_no) {

  switch(item_no) {
  case 0:
    command_table.execute("edit_psdl", home_dir,
                          prototype_name, prototype_ver);
    break;
  case 1:
    command_table.execute("edit_ada", home_dir,
                          prototype_name, prototype_ver);
    break;
  case 2:
    command_table.execute("edit_interface", home_dir,
                          prototype_name, prototype_ver);
    break;
  default:
    break;
  }
}

/*   Callback invoked by the window manager when the Edit option
   is chosen.
*/

static void edit_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;
```

```
  handle_edit_cb(item_no);
}

/*   Handles menu choices from the browse menu, either from the
   menu or from alt-key combinations.
*/

void handle_browse_cb(int item_no) {

  switch(item_no) {
  case 0:
    command_table.execute("browse_ddb", home_dir,
                          prototype_name, prototype_ver);
    break;
  case 1:
    command_table.execute("browse_sb", home_dir,
                          prototype_name, prototype_ver);
    break;
  default:
    break;
  }
}

/*   Callback for the Browse option.
*/

static void browse_cb(Widget, XtPointer client_data,
                      XtPointer) {
  int item_no = (int) client_data;

  handle_browse_cb(item_no);
}

/*   Handles menu choices from either the run menu or from
   appropriate alt-key combinations.
*/

void handle_run_cb(int item_no) {

  switch(item_no) {
  case 0:
    command_table.execute("make", home_dir,
                          prototype_name, prototype_ver);
    break;
  case 1:
    command_table.execute("translate", home_dir,
                          prototype_name, prototype_ver);
    break;
  case 2:
    command_table.execute("compile", home_dir,
                          prototype_name, prototype_ver);
    break;
  case 3:
```

```
    command_table.execute("execute", home_dir,
                          prototype_name, prototype_ver);
    break;
  default:
    break;
  }
}

/*   Run menu callback.
*/

static void run_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;

  handle_run_cb(item_no);
}

/*   Handles choices for the help menu, either from the top
   line menu or from alt-key combinations.
*/

void handle_help_cb(int item_no) {

  switch(item_no) {
  case 0:
    cout << "help selected" << endl;
    break;
  case 1:
    cout << "about selected" << endl;
    break;
  default:
    break;
  }
}

/*   Callback for the Help menu option.
*/

static void help_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;

  handle_help_cb(item_no);
}

/*   Handles choices for the step save option, either from the
   top-line menu or from alt-key combinations.
*/

void handle_save_cb(int item_no) {

  switch(item_no) {
  case 0:
    command_table.execute("save_final", home_dir,
                          prototype_name, prototype_ver);
```

```
    break;
  case 1:
    command_table.execute("suspend", home_dir,
                          prototype_name, prototype_ver);
    break;
  case 2:
    command_table.execute("abandon", home_dir,
                          prototype_name, prototype_ver);
    break;
  default:
    break;
  }
}

/*   Callback for the Step Save menu option.
*/

static void save_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;

  handle_save_cb(item_no);
}

//   Draws the string in the main window.

void draw_logo(Widget widget) {
  XFontStruct *font_ptr;
  Display *display_ptr = XtDisplay(widget);
  Window window = XtWindow(widget);

  XFillRectangle(display_ptr, window, white_context, 0, 0,
                 4000, 2500);

  font_ptr = XLoadQueryFont(display_ptr,
                            "*adobe-times-bold-r*240*");
  XSetFont(display_ptr, black_context, font_ptr->fid);
  XDrawString(display_ptr, window, black_context, 50,
              50, "Computer", strlen("Computer"));
  XDrawString(display_ptr, window, black_context, 50,
              75, "Aided", strlen("Aided"));
  XDrawString(display_ptr, window, black_context, 50,
              100, "Prototyping", strlen("Prototyping"));
  XDrawString(display_ptr, window, black_context, 50,
              125, "System", strlen("System"));
  free((char *) font_ptr);
}

//   Redraws the drawing window when it is resized or exposed.

void redraw(Widget widget, XtPointer , XtPointer cbs) {
  XmDrawingAreaCallbackStruct *temp_ptr;

  temp_ptr = (XmDrawingAreaCallbackStruct *) cbs;
  draw_logo(widget);
```

```
}

//   Creates the top-line menu bar.

void build_menu_bar(Widget &main_w, Widget &menubar) {
  XmString new_step, open, ddb, sb, about, close, save, psdl,
    ada, interface, make, translate, compile, execute, quit,
    browse, step, edit, run, help, new_proto, existing,
    save_final, suspend, abandon;
  Widget widget, step_menu, quit_button,
    new_proto_button, new_exist_button, new_menu, save_menu,
    save_final_button, suspend_button, abandon_button,
    edit_menu, psdl_button, ada_button, interface_button,
    browse_menu, browse_button, ddb_button, sb_button, run_menu,
    make_button, translate_button, compile_button,
    execute_button, help_menu, help1_button, help2_button,
    about_button;

  step = XmStringCreateSimple("Step");
  edit = XmStringCreateSimple("Edit");
  browse = XmStringCreateSimple("Browse");
  run = XmStringCreateSimple("Run");
  help = XmStringCreateSimple("Help");
  quit = XmStringCreateSimple("Quit");
  new_step = XmStringCreateSimple("New");
  open = XmStringCreateSimple("Open");
  ddb = XmStringCreateSimple("Design Data Base");
  sb = XmStringCreateSimple("Software Base");
  about = XmStringCreateSimple("About CAPS");
  help = XmStringCreateSimple("CAPS Help");
  close = XmStringCreateSimple("Close");
  save = XmStringCreateSimple("Save to DDB");
  psdl = XmStringCreateSimple("PSDL");
  ada = XmStringCreateSimple("Ada");
  interface = XmStringCreateSimple("Interface");
  make = XmStringCreateSimple("Make");
  translate = XmStringCreateSimple("Translate");
  compile = XmStringCreateSimple("Compile");
  execute = XmStringCreateSimple("Execute");
  quit = XmStringCreateSimple("Quit");
  new_proto = XmStringCreateSimple("New Prototype");
  existing = XmStringCreateSimple("From Existing Prototype");
  save_final =
    XmStringCreateSimple("Save Approved Final Version");
  suspend =
    XmStringCreateSimple("Suspend Step, May Restart Later");
  abandon =
    XmStringCreateSimple("Abandon Step, No Further Work");
  menubar = XmCreateMenuBar(main_w, "menubar", NULL, 0);
  step_menu =
    XmCreatePulldownMenu(menubar, "step_menu", NULL, 0);
  XtVaCreateManagedWidget("Step", xmCascadeButtonWidgetClass,
                          menubar, XmNlabelString, step,
                          XmNsubMenuId, step_menu,
```

```
                        NULL);
new_menu = XmCreatePulldownMenu(step_menu, "new_menu", NULL,
                                0);
new_button = XtVaCreateManagedWidget("New",
                                xmCascadeButtonWidgetClass,
                                    step_menu,
                                    XmNlabelString, new_step,
                                    XmNsubMenuId, new_menu,
                                    NULL);
new_proto_button = XtVaCreateManagedWidget("new_proto",
                                xmPushButtonGadgetClass,
                                        new_menu, NULL);
XtVaSetValues(new_proto_button, XmNlabelString, new_proto,
              XmNmnemonic, 'N', NULL);
new_exist_button = XtVaCreateManagedWidget("new_exist",
                    xmPushButtonGadgetClass, new_menu, NULL);
XtVaSetValues(new_exist_button, XmNlabelString, existing,
              XmNmnemonic, 'x', NULL);
XtAddCallback(new_proto_button, XmNactivateCallback,
              new_step_cb, (XtPointer) 0);
XtAddCallback(new_exist_button, XmNactivateCallback,
              new_step_cb, (XtPointer) 1);
open_button = XtVaCreateManagedWidget("Open",
                                    xmPushButtonGadgetClass,
                                    step_menu, NULL);
save_menu = XmCreatePulldownMenu(step_menu, "step_menu", NULL,
                                0);
save_button = XtVaCreateManagedWidget("Save",
                                xmCascadeButtonWidgetClass,
                                    step_menu,
                                    XmNlabelString, save,
                                    XmNsubMenuId, save_menu,
                                    NULL);
save_final_button = XtVaCreateManagedWidget("save_final",
                                    xmPushButtonGadgetClass,
                                        save_menu, NULL);
suspend_button = XtVaCreateManagedWidget("suspend",
                                xmPushButtonGadgetClass,
                                    save_menu, NULL);
abandon_button = XtVaCreateManagedWidget("abandon",
                                    xmPushButtonGadgetClass,
                                    save_menu, NULL);
XtVaSetValues(save_final_button, XmNlabelString, save_final,
              XmNmnemonic, 'S', NULL);
XtVaSetValues(suspend_button, XmNlabelString, suspend,
              XmNmnemonic, 'u', NULL);
XtVaSetValues(abandon_button, XmNlabelString, abandon,
              XmNmnemonic, 'A', NULL);
XtAddCallback(save_final_button, XmNactivateCallback,
              save_cb, (XtPointer) 0);
XtAddCallback(suspend_button, XmNactivateCallback, save_cb,
              (XtPointer) 1);
XtAddCallback(abandon_button, XmNactivateCallback, save_cb,
              (XtPointer) 2);
```

```
close_button = XtVaCreateManagedWidget("Close",
                                       xmPushButtonGadgetClass,
                                          step_menu, NULL);
quit_button = XtVaCreateManagedWidget("Quit",
                                         xmPushButtonGadgetClass,
                                         step_menu, NULL);
XtVaSetValues(open_button, XmNmnemonic, 'O', NULL);
XtVaSetValues(close_button, XmNmnemonic, 'C', NULL);
XtVaSetValues(quit_button, XmNmnemonic, 'Q', NULL);

XtAddCallback(open_button, XmNactivateCallback, step_cb,
              (XtPointer) 1);
XtAddCallback(close_button, XmNactivateCallback, step_cb,
              (XtPointer) 3);
XtAddCallback(quit_button, XmNactivateCallback, step_cb,
              (XtPointer) 4);

edit_menu = XmCreatePulldownMenu(menubar, "edit_menu", NULL,
                                    0);
edit_button = XtVaCreateManagedWidget("Edit",
                                     xmCascadeButtonWidgetClass,
                                         menubar,
                                         XmNlabelString, edit,
                                         XmNsubMenuId, edit_menu,
                                         NULL);
psdl_button = XtVaCreateManagedWidget("PSDL",
                                         xmPushButtonGadgetClass,
                                         edit_menu,
                                         XmNlabelString, psdl,
                                         XmNmnemonic, 'P', NULL);
ada_button = XtVaCreateManagedWidget("Ada",
                                       xmPushButtonGadgetClass,
                                       edit_menu,
                                       XmNlabelString, ada,
                                       XmNmnemonic, 'd', NULL);
interface_button = XtVaCreateManagedWidget("Interface",
                                     xmPushButtonGadgetClass,
                                             edit_menu,
                                             XmNlabelString,
                                             interface,
                                             XmNmnemonic, 'I',
                                             NULL);
XtAddCallback(psdl_button, XmNactivateCallback, edit_cb,
              (XtPointer) 0);
XtAddCallback(ada_button, XmNactivateCallback, edit_cb,
              (XtPointer) 1);
XtAddCallback(interface_button, XmNactivateCallback, edit_cb,
              (XtPointer) 2);

browse_menu = XmCreatePulldownMenu(menubar, "browse_menu",
                                     NULL, 0);
browse_button = XtVaCreateManagedWidget("Browse",
                                     xmCascadeButtonWidgetClass,
                                     menubar,
```

```
                                    XmNlabelString, browse,
                                    XmNsubMenuId,
                                    browse_menu,
                                    NULL);
ddb_button = XtVaCreateManagedWidget("DDB",
                                      xmPushButtonGadgetClass,
                                      browse_menu,
                                      XmNlabelString, ddb,
                                      XmNmnemonic, 'g', NULL);
sb_button = XtVaCreateManagedWidget("SB",
                                    xmPushButtonGadgetClass,
                                    browse_menu,
                                    XmNlabelString, sb,
                                    XmNmnemonic, 'B', NULL);
XtAddCallback(ddb_button, XmNactivateCallback, browse_cb,
              (XtPointer) 0);
XtAddCallback(sb_button, XmNactivateCallback, browse_cb,
              (XtPointer) 1);

run_menu = XmCreatePulldownMenu(menubar, "run_menu", NULL, 0);
run_button = XtVaCreateManagedWidget("Run",
                                    xmCascadeButtonWidgetClass,
                menubar,
                XmNlabelString, run,
                XmNsubMenuId, run_menu,
                NULL);
make_button = XtVaCreateManagedWidget("Make",
                                      xmPushButtonGadgetClass,
                                      run_menu,
                                      XmNlabelString, make,
                                      XmNmnemonic, 'M', NULL);
translate_button = XtVaCreateManagedWidget("Translate",
                            xmPushButtonGadgetClass, run_menu,
                                    XmNlabelString, translate,
                                    XmNmnemonic, 'T', NULL);
compile_button = XtVaCreateManagedWidget("Compile",
                                      xmPushButtonGadgetClass,
  run_menu,
  XmNlabelString, compile,
  XmNmnemonic, 'l', NULL);
execute_button = XtVaCreateManagedWidget("Execute",
                                    xmPushButtonGadgetClass,
                                        run_menu,
                                    XmNlabelString, execute,
                                    XmNmnemonic, 'E', NULL);
XtAddCallback(make_button, XmNactivateCallback, run_cb,
              (XtPointer) 0);
XtAddCallback(translate_button, XmNactivateCallback, run_cb,
              (XtPointer) 1);
XtAddCallback(compile_button, XmNactivateCallback, run_cb,
              (XtPointer) 2);
XtAddCallback(execute_button, XmNactivateCallback, run_cb,
              (XtPointer) 3);
```

```
  help_menu = XmCreatePulldownMenu(menubar, "help_menu", NULL,
                                   0);
  help1_button = XtVaCreateManagedWidget("Help",
                                   xmCascadeButtonWidgetClass,
                                      menubar,
                                      XmNlabelString, help,
                                      XmNsubMenuId, help_menu,
                                      NULL);
  help2_button = XtVaCreateManagedWidget("Help_but",
                                      xmPushButtonGadgetClass,
                                      help_menu,
                                      XmNlabelString, help,
                                      XmNmnemonic, 'H', NULL);
  about_button = XtVaCreateManagedWidget("About",
                                      xmPushButtonGadgetClass,
                                      help_menu,
                                      XmNlabelString, about,
                                      NULL);
  XtAddCallback(help2_button, XmNactivateCallback, help_cb,
                (XtPointer) 0);
  XtAddCallback(about_button, XmNactivateCallback, help_cb,
                (XtPointer) 1);

  if(widget = XtNameToWidget(menubar, "Help"))
    XtVaSetValues(menubar, XmNmenuHelpWidget, widget, NULL);

  XmStringFree(new_step);
  XmStringFree(open);
  XmStringFree(ddb);
  XmStringFree(sb);
  XmStringFree(about);
  XmStringFree(help);
  XmStringFree(close);
  XmStringFree(psdl);
  XmStringFree(ada);
  XmStringFree(interface);
  XmStringFree(make);
  XmStringFree(translate);
  XmStringFree(compile);
  XmStringFree(execute);
  XmStringFree(quit);
  XmStringFree(save);
  XmStringFree(edit);
  XmStringFree(browse);
  XmStringFree(run);
  XmStringFree(quit);
  XmStringFree(help);
  XmStringFree(new_proto);
  XmStringFree(existing);
  XmStringFree(save_final);
  XmStringFree(suspend);
  XmStringFree(abandon);

}
```

```
//   Handles the keyboard accelerators.

void draw(Widget, XEvent *event, String *args, Cardinal *) {

  if(strcmp(args[0], "key") == 0) {
    if(alt_key_selected) {
      alt_key_selected = FALSE;
      switch(event->xkey.keycode) {
      case NEW_STEP:
        handle_new_step_cb(0);
        break;
      case NEW_EXISTING:
        handle_new_step_cb(1);
        break;
      case OPEN:
        handle_step_cb(1);
        break;
      case SAVE_FINAL:
        handle_save_cb(0);
        break;
      case SUSPEND:
        handle_save_cb(1);
        break;
      case ABANDON:
        handle_save_cb(2);
        break;
      case CLOSE:
        handle_step_cb(3);
        break;
      case QUIT:
        handle_step_cb(4);
        break;
      case EDIT_PSDL:
        handle_edit_cb(0);
        break;
      case EDIT_ADA:
        handle_edit_cb(1);
        break;
      case EDIT_INTERFACE:
        handle_edit_cb(2);
        break;
      case BROWSE_DDB:
        handle_browse_cb(0);
        break;
      case BROWSE_SB:
        handle_browse_cb(1);
        break;
      case MAKE:
        handle_run_cb(0);
        break;
      case TRANSLATE:
        handle_run_cb(1);
        break;
```

```
      case COMPILE:
        handle_run_cb(2);
        break;
      case EXECUTE:
        handle_run_cb(3);
        break;
      case HELP:
        handle_help_cb(0);
        break;
      case ALT:
        alt_key_selected = TRUE;
        break;
      default:
        cout << event->xkey.keycode << endl;
        break;
      }
    }
    else
      if(event->xkey.keycode == ALT)
        alt_key_selected = TRUE;
  }
}


int main(unsigned int argc, char **argv) {
  Widget toplevel, menubar, main_w, drawing_a;
  XtAppContext app;
  Screen *screen_ptr;
  Display *display_ptr;
  XGCValues gcv;
  unsigned long gc_mask;
  XtActionsRec actions;
  String translations =
    "<Key>:         draw(key)\n\
     <Key>Tab:    draw(tab)";

  toplevel = XtVaAppInitialize(&app, "CAPS_Menu", NULL, 0,
                               &argc, argv,
                               NULL, NULL);
  main_w = XtVaCreateManagedWidget("main_w", xmFormWidgetClass,
                                   toplevel, NULL);
  init();
  main_window = main_w;
  build_menu_bar(main_w, menubar);
  XtManageChild(menubar);
  set_step_state_false();
  drawing_a = XtVaCreateManagedWidget("drawing_a",
                                      xmDrawingAreaWidgetClass,
                                      main_w,
                                    XmNunitType, Xm1000TH_INCHES,
                                      XmNwidth, 4000,
                                      XmNheight, 2500,
                                      XmNresizePolicy, XmNONE,
                                      NULL);
```

```
    XtVaSetValues(drawing_a,  XmNtranslations,
                  XtParseTranslationTable(translations));
    XtVaSetValues(menubar, XmNtopAttachment, XmATTACH_FORM,
                  XmNrightAttachment, XmATTACH_FORM,
                  XmNleftAttachment, XmATTACH_FORM,
                  XmNbottomAttachment, XmATTACH_NONE,
                  NULL);
    XtVaSetValues(drawing_a, XmNtopAttachment, XmATTACH_WIDGET,
                  XmNtopWidget, menubar,
                  XmNrightAttachment, XmATTACH_FORM,
                  XmNleftAttachment, XmATTACH_FORM,
                  XmNbottomAttachment, XmATTACH_FORM,
                  NULL);

  XtVaSetValues(drawing_a, XmNunitType, XmPIXELS, NULL);
  XtAddCallback(drawing_a, XmNexposeCallback, redraw, NULL);
  screen_ptr = XtScreen(toplevel);
  display_ptr = XtDisplay(toplevel);
  gcv.foreground = BlackPixelOfScreen(XtScreen(toplevel));
  gcv.background = WhitePixelOfScreen(XtScreen(toplevel));
  gc_mask = GCForeground | GCBackground;
  black_context = XCreateGC(display_ptr,
                            RootWindowOfScreen(screen_ptr),
                            gc_mask, &gcv);
  gcv.foreground = WhitePixelOfScreen(XtScreen(toplevel));
  white_context = XCreateGC(display_ptr,
                            RootWindowOfScreen(screen_ptr),
                            gc_mask, &gcv);
  XtVaSetValues(toplevel, XmNtitle, "CAPS 93", NULL);
  actions.string = "draw";
  actions.proc = draw;
  XtAppAddActions(app, &actions, 1);

  XtRealizeWidget(toplevel);
  draw_logo(drawing_a);
  XFlush(display_ptr);
  cout << "\n\n" << endl;
  XmProcessTraversal(menubar, XmTRAVERSE_CURRENT);
  XtAppMainLoop(app);
  return 0;
}
```

```
/* ************************************************************

Name:           strstack.h
Author:         Capt Robert M. Dixon
Program:        caps93
Date Modified:  19 Sep 92
Remarks:          Implements a stack and linked list for storing
                strings.

************************************************************ */
#ifndef _strstack_h
#define _strstack_h

#include <string.h>

enum boolean {false, true};

class strstack;

class strnode {

protected:
  char *str;
  strnode *next;
public:
  strnode(char *);
  ~strnode();
  void link(strnode *nextnode) {next = nextnode;};
  void unlink() {next = NULL;};
  friend class strstack;
  friend class strlist;
};

class strstack {

  strnode *head;
public:
  strstack() {head = NULL;};
  ~strstack() {delete head;};
  void push(char *newstring);
  char *pop();
  const char *top() {return head->str;};
  boolean notempty();
};

class strlist {

  strnode *head;
public:
  strlist() {head = NULL;};
  ~strlist() {delete head;};
  void add(char *newstring);
  char *remove();
  boolean notempty();
```

```
};

#endif
```

```
/* ************************************************************

Name:           strstack.C
Author:         Capt Robert M. Dixon
Program:        caps93
Date Modified: 19 Sep 92
Remarks:   Implements a stack and linked list for strings.
         The list is stored in reverse alphabetical order so that
         when it is later pushed on a stack it will be in the
         correct order on the stack. These classes are
         specifically for directory traversals.


************************************************************* */
#include "strstack.h"

strnode::strnode(char *newstring) {

  str = new char[strlen(newstring) + 1];
  strcpy(str, newstring);
  next = NULL;
}

strnode::~strnode() {

  delete str;
  delete next;
}

void strstack::push(char *newstring) {
  strnode *newnode = new strnode(newstring);

  newnode->link(head);
  head = newnode;
}

char *strstack::pop() {
  strnode *topnode = head;

  if(notempty()) {
    char *outstring = new char[strlen(head->str) + 1];
    strcpy(outstring, head->str);
    head = head->next;
    topnode->unlink();
    delete topnode;
    return outstring;
  }
}

//   Returns true if the stack is not empty.

boolean strstack::notempty() {

  if(head != NULL)
```

```
    return true;
  else
    return false;
}

//   Adds a new node to the linked list. Stores in reverse
// alphabetical order.

void strlist::add(char *newstring) {
  strnode *newnode = new strnode(newstring);
  strnode *prev, *curr;

  if(notempty()) {
    if(strcmp(head->str, newnode->str) > 0) {
      newnode->link(head);
      head = newnode;
    }
    else {
      prev = head;
      curr = head->next;
      while(curr != NULL) {
        if(strcmp(curr->str, newnode->str) > 0)
          break;
        prev = curr;
        curr = curr->next;
      }
      newnode->link(curr);
      prev->link(newnode);
    }
  }
  else
    head = newnode;
}

//   Removes the top node and returns the string.

char *strlist::remove() {
  char *outstring = new char[strlen(head->str) + 1];
  strnode *topnode = head;

  strcpy(outstring, head->str);
  head = head->next;
  topnode->unlink();
  delete topnode;
  return outstring;
}

//   Returns TRUE if the list is not empty.

boolean strlist::notempty() {

  if(head != NULL)
    return true;
  else
```

```
    return false;
}
```

```
/* ************************************************************

Name:           extern.h
Author:         Capt Robert M. Dixon
Program:        sde
Date Modified: 12 Sep 92
Remarks:   Provides the external variable declarations for
         the syntax-directed editor and the interface
         routines.

           Originally created by Professor Valdis Berzins and
         modified by Professor Fernando Naveda and Capt Robert
         M. Dixon.


************************************************************* */
#ifndef EXTERN_H
#define EXTERN_H 1

#include "sde_ge.h"

extern OPNodePTR the_operator_list;
extern ST_PTR the_stream_list;
extern HeadPtr prototype;
extern HeadPtr current_graph;
extern int op_id_count;
extern int level_change_direction;
extern char *goto_child;

#endif
```

```
/* ************************************************************

Name:           font_table.h
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 12 Sep 92
Remarks:   Specification for the FontTable object.

          The FontTable contains all the necessary information
        about the fonts used in the graph editor, allowing
        the program to refer to them by a font id#.

          It initializes with six predefined font names,
        although more could easily be added.

************************************************************* */
#ifndef FONT_TABLE_H
#define FONT_TABLE_H 1

#include <stdlib.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <Xm/MessageB.h>
#include <string.h>
#include "ge_defs.h"

struct _font_record {
  char *_name_ptr;
  XFontStruct  *_font_ptr;
  int _height;
};

class FontTable {
private:
  struct _font_record _font_table[MAXFONTS + 1];
  static Widget _error_tgt;

public:
  static void set_error_tgt(Widget widget) {_error_tgt = widget;}
  static void error_box(char *error_message);

  FontTable();
  ~FontTable();
  void init(Display *display_ptr);
  int width(int font_id, char *in_string);
  int height(int font_id);
  Font font_id(int font_id);
  char *font_name(int font_id);
};

#endif
```

```
/* ************************************************************

Name:
Author:         Capt Robert M. Dixon
Program:
Date Modified:
Remarks:    Implementation of the FontTable object.

            The FontTable contains all the necessary information
          about the fonts used in the graph editor, allowing
          the program to refer to them by a font id#.

            It initializes with six predefined font names,
          although more could easily be added.


Credits:      Portions of code are adapted from the following:

            Heller, Dan, Motif Programming Manual, O'Reilly and
              Associates, 1991.

            Johnson, Eric, and Reichard, Kevin, X Window
              Applications Programming, MIS Press, 1989.

            Young, Douglas, Object Oriented Programming With C++
              and OSF/Motif, Prentice-Hall, 1992.

************************************************************* */
#include <stream.h>
#include "font_table.h"

//   Initializes static class variable.

Widget FontTable::_error_tgt = NULL;

//   Displays the error message in a Motif error dialog box.

void FontTable::error_box(char *error_message) {
  static Widget error_dialog = NULL;
  Arg arg[1];
  XmString t;

  if(_error_tgt != NULL) {
    if(error_dialog == NULL) {
      error_dialog = XmCreateMessageDialog(_error_tgt, "error",
                                           arg, 0);
      XtVaSetValues(XtParent(error_dialog),
                    XtNtitle, "Error",
                    NULL);
      XtUnmanageChild(XmMessageBoxGetChild(error_dialog,
                                           XmDIALOG_HELP_BUTTON));
    }
    t = XmStringCreateSimple(error_message);
```

```
    XtVaSetValues(error_dialog,
                  XmNmessageString, t,
                  NULL);
    XmStringFree(t);
    XtManageChild(error_dialog);
  }
  else
    cout <<
      "Error Target Widget must be set by calling module.\n"
        << endl;
}

FontTable::FontTable() {
  int i;

  for(i = 1; i <= MAXFONTS; i++) {
    _font_table[i]._name_ptr = NULL;
    _font_table[i]._font_ptr = NULL;
  }
}


FontTable::~FontTable() {
  int i;

  for(i = 1; i <= MAXFONTS; i++) {
    delete _font_table[i]._name_ptr;
    delete _font_table[i]._font_ptr;
  }
}

//   Initializes the font table using fonts defined for the
// given display.

void FontTable::init(Display *display_ptr) {

  _font_table[0]._name_ptr =
                        strdup("variable");
  _font_table[COURIERBOLD10]._name_ptr =
                        strdup("*adobe-courier-bold-r*100*");
  _font_table[COURIERBOLD12]._name_ptr =
                        strdup("*adobe-courier-bold-r*120*");
  _font_table[COURIERBOLD14]._name_ptr =
                        strdup("*adobe-courier-bold-r*140*");
  _font_table[COURIERMED10]._name_ptr =
                        strdup("*adobe-courier-medium-r*100*");
  _font_table[COURIERMED12]._name_ptr =
                        strdup("*adobe-courier-medium-r*120*");
  _font_table[COURIERMED14]._name_ptr =
                        strdup("*adobe-courier-medium-r*140*");

  _font_table[0]._font_ptr =
    XLoadQueryFont(display_ptr, _font_table[0]._name_ptr);
  _font_table[COURIERBOLD10]._font_ptr =
```

```
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERBOLD10]._name_ptr);
  _font_table[COURIERBOLD12]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERBOLD12]._name_ptr);
  _font_table[COURIERBOLD14]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERBOLD14]._name_ptr);
  _font_table[COURIERMED10]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERMED10]._name_ptr);
  _font_table[COURIERMED12]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERMED12]._name_ptr);
  _font_table[COURIERMED14]._font_ptr =
    XLoadQueryFont(display_ptr,
                   _font_table[COURIERMED14]._name_ptr);

  _font_table[0]._height =
    _font_table[0]._font_ptr->ascent +
      _font_table[0]._font_ptr->descent;

  _font_table[COURIERBOLD10]._height =
    _font_table[COURIERBOLD10]._font_ptr->ascent +
      _font_table[COURIERBOLD10]._font_ptr->descent;

  _font_table[COURIERBOLD12]._height =
    _font_table[COURIERBOLD12]._font_ptr->ascent +
      _font_table[COURIERBOLD12]._font_ptr->descent;

  _font_table[COURIERBOLD14]._height =
    _font_table[COURIERBOLD14]._font_ptr->ascent +
      _font_table[COURIERBOLD14]._font_ptr->descent;

  _font_table[COURIERMED10]._height =
    _font_table[COURIERMED10]._font_ptr->ascent +
      _font_table[COURIERMED10]._font_ptr->descent;

  _font_table[COURIERMED12]._height =
    _font_table[COURIERMED12]._font_ptr->ascent +
      _font_table[COURIERMED12]._font_ptr->descent;

  _font_table[COURIERMED14]._height =
    _font_table[COURIERMED14]._font_ptr->ascent +
      _font_table[COURIERMED14]._font_ptr->descent;
}

//   Returns the width of the given string in the given font
// in pixels.

int FontTable::width(int font_id, char *in_string) {

  if(in_string == NULL)
    return 0;
```

```
  else
    if(font_id > MAXFONTS)
      error_box("Font Table entered with font too big.");
    else
      return XTextWidth(_font_table[font_id]._font_ptr,
                        in_string, strlen(in_string));
}

//   Returns the height of the given font in pixels.

int  FontTable::height(int font_id) {

  if(font_id <= MAXFONTS)
    return _font_table[font_id]._height;
  else {
    error_box("Font Table entered with font too big.");
    return 0;
  }
}

//   Returns font information needed by some X functions.

Font FontTable::font_id(int font_id) {

  if(font_id <= MAXFONTS)
    return _font_table[font_id]._font_ptr->fid;
  else {
    error_box("Font Table entered with font too big.");
    return 0;
  }
}

//   Returns the name of the given font.

char *FontTable::font_name(int font_id) {

  if(font_id <= MAXFONTS)
    return _font_table[font_id]._name_ptr;
  else {
    error_box("Font Table entered with font too big.");
    return NULL;
  }
}
```

```
/* ************************************************************

Name:           ge_defs.h
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 12 Sep 92
Remarks:    Provides the common type definitions and defines
          for the modules in the graph_editor program.


************************************************************ */
#ifndef ge_defs_h
#define ge_defs_h 1

typedef int BOOLEAN;
enum CLASS_DEF{GRAPHOBJECT, OPERATOROBJECT, STREAMOBJECT,
               GRAPHOBJECTLIST, SPLINEOBJECT};

enum TOOL_STATE{OPERATOR_TOOL, TERMINATOR_TOOL, STREAM_TOOL,
                MET_TOOL, LATENCY_TOOL, SELECT_TOOL};

enum DRAW_STYLE{SOLID, DOTTED, ERASE};

typedef int GE_STATUS;
typedef int EXTERN_STATUS;
typedef int COLOR;

typedef struct xypair {
  int x, y;
} XYPAIR;

#define TRUE 1
#define FALSE 0
#define ENDED 2
#define SUCCEEDED 1
#define FAILED 0
#define INPUT_LINE_SIZE 100
#define CIRCLE_BEGIN 0
#define FULL_CIRCLE 360 * 64
#define HANDLESIZE 5

/*   Return codes used to tell the syntax-directed editor
  what to do with the incoming data, and whether any
  level changes are required.
*/

#define UP -1
#define SAME 0
#define ERROR -3
#define NOUPDATE -4

#define NO_EXTERNAL 0
#define FROM_EXTERNAL 1
#define TO_EXTERNAL 2
```

```
#define HITFUDGE 5

#define NULL_VALUE 0
#define MAXDELETEDOPS 100
#define MAXTEXTLINES 100

#include "resources.h"
#endif
```

```
/* ************************************************************

Name:           ge_interface.c
Author:         Capt Robert M. Dixon
Program:        sde
Date Modified: 12 Sep 92
Remarks:   ge_interface.c contains all the interface routines
          to allow the syntax-directed editor to communicate
          with the graph editor.

            The syntax-directed editor maintains global variables
          and data structures that define the graph. It calls
          the functions in ge_interface.c, which perform
          various operations on those data structures.

Credits:     Portions of code are adapted from the following:

           Barakati, Naba, X Window System Programming, SAMS,
             1991.

           Heller, Dan, Motif Programming Manual, O'Reilly and
             Associates, 1991.

           Johnson, Eric, and Reichard, Kevin, X Window
             Applications Programming, MIS Press, 1989.

           Young, Douglas, Object Oriented Programming With C++
             and OSF/Motif, Prentice-Hall, 1992.


************************************************************* */
#include <stdlib.h>
#include <stdio.h>
#include <strings.h>
#include <X11/Xatom.h>
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Intrinsic.h>
#include <Xm/Xm.h>
#include <Xm/Label.h>
#include <pwd.h>
#include "extern.h"
#include "resources.h"

extern char * strdup();

#define SUCCEEDED 1
#define FAILED 0
#define INPUT_LINE_SIZE 100
#define MAX_OPERATORS 50
#define MAX_STREAMS 50
#define MAX_SPLINE_NODES 50
#define NULL_VALUE 0     /* If changed, change in
                            graph_editor.C also. */
```

```
/*   These codes are returned by the graph editor to tell
  the syntax-directed editor whether a change in levels is
  required, or whether to ignore any changes to the graph.
*/

#define GRUP -1
#define SAME 0
#define ERROR -3
#define NOUPDATE -4

typedef int STATUS;

/*   Displays the contents of the operator node list
  pointed to by in_ptr.
*/

void  disp_ge_opnodes(in_ptr)
     struct op_node *in_ptr;
{
  struct op_node *temp_node_ptr = in_ptr;

  while(temp_node_ptr != NULL) {
    if(temp_node_ptr->op->name == NULL)
       printf("Operator Name == NULL\n");
    else
      printf("Name: %s ", temp_node_ptr->op->name);
    printf("Font: %d %d %d ", temp_node_ptr->op->name_font,
          temp_node_ptr->op->name_x, temp_node_ptr->op->name_y);
    printf("Id: %d ", temp_node_ptr->op->id);
    if(temp_node_ptr->op->met == NULL)
      printf("Met: None ");
    else
      printf("Met: %d ", *(temp_node_ptr->op->met));
    printf("Met Font: %d %d %d ", temp_node_ptr->op->met_font,
          temp_node_ptr->op->met_x, temp_node_ptr->op->met_y);
    printf("x: %d ", temp_node_ptr->op->x);
    printf("y: %d ", temp_node_ptr->op->y);
    printf("radius: %d ", temp_node_ptr->op->radius);
    printf("color: %d ", temp_node_ptr->op->color);
    if(temp_node_ptr->op->is_deleted)
      printf("     deleted ");
    else
      printf(" not_deleted ");
    if(temp_node_ptr->op->is_new)
      printf("     new ");
    else
      printf(" not_new ");
    if(temp_node_ptr->op->is_modified)
      printf("     modified ");
    else
      printf(" not_modified ");
    if(temp_node_ptr->op->is_composite == NULL)
      printf(
```

```
            "is_composite pointer is NULL. Possible problem.\n");
    else {
      if(*(temp_node_ptr->op->is_composite))
        printf("     composite ");
      else
        printf(" not_composite ");
    }
    if(temp_node_ptr->op->is_terminator)
      printf("     terminator \n");
    else
      printf(" not_terminator \n");
    temp_node_ptr = temp_node_ptr->next;
  }
}

/*   Displays the contents of the stream node list pointed to
  by in_ptr.
*/

void  disp_ge_stnodes(in_ptr)
     struct st_node *in_ptr;
{
  struct st_node *temp_node_ptr = in_ptr;
  struct spline_node *temp_spline_ptr;

  while(temp_node_ptr != NULL) {
    if(temp_node_ptr->st->name == NULL)
      printf("Stream name == NULL");
    else
      printf("Name: %s ", temp_node_ptr->st->name);
    printf("Font: %d %d %d ", temp_node_ptr->st->name_font,
           temp_node_ptr->st->name_x,
           temp_node_ptr->st->name_y);
    printf("Id: %d ", temp_node_ptr->st->id);
    printf("From: ");
    if(temp_node_ptr->st->from != NULL)
      printf(" %d ", temp_node_ptr->st->from->op->id);
    else
      printf("0");
    printf("To: ");
    if(temp_node_ptr->st->to != NULL)
      printf("%d ", temp_node_ptr->st->to->op->id);
    else
      printf("0");
    printf("{");
    temp_spline_ptr = temp_node_ptr->st->arc;
    while(temp_spline_ptr != NULL) {
      printf(" %d %d,", temp_spline_ptr->x, temp_spline_ptr->y);
      temp_spline_ptr = temp_spline_ptr->next;
    }
    printf("}");
    printf("Latency: %d ", temp_node_ptr->st->latency);
    printf("Latency Font: %d x: %d y: %d ",
           temp_node_ptr->st->latency_font,
```

```
            temp_node_ptr->st->latency_x,
            temp_node_ptr->st->latency_y);
    if(temp_node_ptr->st->is_deleted)
      printf("     deleted ");
    else
      printf(" not_deleted ");
    if(temp_node_ptr->st->is_new)
      printf("     new ");
    else
      printf(" not_new ");
    if(temp_node_ptr->st->is_modified)
      printf("     modified ");
    else
      printf(" not_modified ");
    if(temp_node_ptr->st->is_state_variable)
      printf("     state_variable \n");
    else
      printf(" not_state_variable \n");
    temp_node_ptr = temp_node_ptr->next;
  }
}

/*   Determines whether the string represents a valid number.
*/

BOOL valid_num_string(num)
     char *num;
{
  int index, num_length;

  if(num != NULL) {
    num_length = strlen(num);
    if(num_length > 0) {
      for(index = 0; index < num_length; index++) {
        if((num[index] != '\n') &&
           (num[index] != '-') &&
           ((num[index] < '0') || (num[index] > '9')))
          return FALSE;
      }
      return TRUE;
    }
  }
  return FALSE;
}

/*   Writes the given operator out to the given output stream.
*/

STATUS write_op_to_file(outfile, out_node)
     FILE *outfile;
     struct op_node* out_node;
{
  char buffer[INPUT_LINE_SIZE + 1];
```

```
if(out_node->op->name == NULL)
  printf("Operator name == NULL in write_op_to_file\n");
else
  fprintf(outfile, "%s\n", out_node->op->name);

sprintf(buffer, "%d", out_node->op->name_font);
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->name_x);
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->name_y);
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->id);
fprintf(outfile, "%s\n", buffer);
if(out_node->op->met == NULL)
  sprintf(buffer, "%d", NULL_VALUE);
else
  sprintf(buffer, "%d", *(out_node->op->met));
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->met_font);
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->met_x);
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->met_y);
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->x);
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->y);
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->radius);
fprintf(outfile, "%s\n", buffer);
sprintf(buffer, "%d", out_node->op->color);
fprintf(outfile, "%s\n", buffer);
if(out_node->op->is_deleted)
  fprintf(outfile, "TRUE\n");
else
  fprintf(outfile, "FALSE\n");
if(out_node->op->is_new)
  fprintf(outfile, "TRUE\n");
else
  fprintf(outfile, "FALSE\n");
if(out_node->op->is_modified)
  fprintf(outfile, "TRUE\n");
else
  fprintf(outfile, "FALSE\n");
if(out_node->op->is_composite == NULL)
  printf("NULL value for is_composite in write_op_to_file.\n");
else {
  if(*(out_node->op->is_composite))
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
}
if(out_node->op->is_terminator)
  fprintf(outfile, "TRUE\n");
```

```
  else
    fprintf(outfile, "FALSE\n");
  if(ferror(outfile)) {
    clearerr(outfile);
    return FAILED;
  }
  return SUCCEEDED;
}

/*   Writes the given stream out to the given disk file.
*/

STATUS write_st_to_file(outfile, out_node)
     FILE *outfile;
     struct st_node* out_node;
{
  char buffer[INPUT_LINE_SIZE + 1];
  struct spline_node* temp_node;

  if(out_node->st->name == NULL)
    printf("Stream name == NULL in write_st_to_file");
  else
    fprintf(outfile, "%s\n", out_node->st->name);
  sprintf(buffer, "%d", out_node->st->name_font);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", out_node->st->name_x);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", out_node->st->name_y);
  fprintf(outfile, "%s\n", buffer);

  sprintf(buffer, "%d", out_node->st->id);
  fprintf(outfile, "%s\n", buffer);
  if(out_node->st->from != NULL)
    sprintf(buffer, "%d", out_node->st->from->op->id);
  else
    sprintf(buffer, "%d", 0);
  fprintf(outfile, "%s\n", buffer);
  if(out_node->st->to != NULL)
    sprintf(buffer, "%d", out_node->st->to->op->id);
  else
    sprintf(buffer, "%d", 0);
  fprintf(outfile, "%s\n", buffer);
  fprintf(outfile, "SPLINE\n");
  temp_node = out_node->st->arc;
  while(temp_node != NULL) {
    sprintf(buffer, "%d", temp_node->x);
    fprintf(outfile, "%s\n", buffer);
    sprintf(buffer, "%d", temp_node->y);
    fprintf(outfile, "%s\n", buffer);
    temp_node = temp_node->next;
  }
  fprintf(outfile, "SPLINEEND\n");
  sprintf(buffer, "%d", out_node->st->latency);
  fprintf(outfile, "%s\n", buffer);
```

```
  sprintf(buffer, "%d", out_node->st->latency_font);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", out_node->st->latency_x);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", out_node->st->latency_y);
  fprintf(outfile, "%s\n", buffer);
  if(out_node->st->is_deleted)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(out_node->st->is_new)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(out_node->st->is_modified)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(out_node->st->is_state_variable)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(ferror(outfile)) {
    clearerr(outfile);
    return FAILED;
  }
  return SUCCEEDED;
}

/*   Concatenates two strings together.
*/

void add_string(instring, new_string)
     char **instring;
     char *new_string;
{
  char *temp_str_ptr;

  temp_str_ptr = (char *) malloc((strlen(*instring) + strlen(new_string) + 1)
                                 * sizeof(char));
  strcpy(temp_str_ptr, *instring);
  strcat(temp_str_ptr, new_string);
  *instring = temp_str_ptr;
}

/*   Builds an operator from disk.
*/

struct op_node* build_op_node_fm_disk(infile, instatus)
     FILE *infile;
     Status *instatus;
{
  char buffer[INPUT_LINE_SIZE + 1];
  int last_char, met_value, temp_int, error_length;
```

```
struct op_node* temp_op_ptr =
         (struct op_node *) malloc(sizeof(struct op_node));
char *error_str_ptr = NULL;
Status status = SUCCEEDED;

temp_op_ptr->op =
              (struct op_str *) malloc(sizeof(struct op_str));
temp_op_ptr->next = NULL;
fgets(buffer, INPUT_LINE_SIZE, infile);
last_char = strlen(buffer) - 1;
if(buffer[last_char] == '\n')
  buffer[last_char] = 0;
if(strcmp(buffer, "STREAMS") == 0) {
  free(temp_op_ptr->op);
  free(temp_op_ptr);
  return NULL;
}
else {
  temp_op_ptr->op->name = strdup(buffer);
  error_str_ptr = strdup(buffer);
  add_string(&error_str_ptr, "elements in error: ");
  error_length = strlen(error_str_ptr);

  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer)) {
    temp_op_ptr->op->name_font = atoi(buffer);
    if(temp_op_ptr->op->name_font > MAXFONTS) {
      add_string(&error_str_ptr, "name_font ");
      temp_op_ptr->op->name_font = MAXFONTS;
    }
  }
  else
    add_string(&error_str_ptr, "name_font ");

  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    temp_op_ptr->op->name_x = atoi(buffer);
  else
    add_string(&error_str_ptr, "name_x ");

  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    temp_op_ptr->op->name_y = atoi(buffer);
  else
    add_string(&error_str_ptr, "name_y ");

  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    temp_op_ptr->op->id = atoi(buffer);
  else
    add_string(&error_str_ptr, "id ");

  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
```

```
  met_value = atoi(buffer);
else {
  met_value = NULL_VALUE;
  add_string(&error_str_ptr, "met ");
}

if(met_value == NULL_VALUE)
  temp_op_ptr->op->met = NULL;
else {
  temp_op_ptr->op->met = (int *) malloc(sizeof(int));
  *(temp_op_ptr->op->met) = met_value;
}

fgets(buffer, INPUT_LINE_SIZE, infile);
if(valid_num_string(buffer)) {
  temp_op_ptr->op->met_font = atoi(buffer);
  if(temp_op_ptr->op->met_font > MAXFONTS) {
    add_string(&error_str_ptr, "met_font ");
    temp_op_ptr->op->met_font = MAXFONTS;
  }
}
else
  add_string(&error_str_ptr, "met_font ");

fgets(buffer, INPUT_LINE_SIZE, infile);
if(valid_num_string(buffer))
  temp_op_ptr->op->met_x = atoi(buffer);
else
  add_string(&error_str_ptr, "met_x ");

fgets(buffer, INPUT_LINE_SIZE, infile);
if(valid_num_string(buffer))
  temp_op_ptr->op->met_y = atoi(buffer);
else
  add_string(&error_str_ptr, "met_y ");

fgets(buffer, INPUT_LINE_SIZE, infile);
if(valid_num_string(buffer))
  temp_op_ptr->op->x = atoi(buffer);
else
  add_string(&error_str_ptr, "x ");

fgets(buffer, INPUT_LINE_SIZE, infile);
if(valid_num_string(buffer))
  temp_op_ptr->op->y = atoi(buffer);
else
  add_string(&error_str_ptr, "y ");

fgets(buffer, INPUT_LINE_SIZE, infile);
if(valid_num_string(buffer))
  temp_op_ptr->op->radius = atoi(buffer);
else
  add_string(&error_str_ptr, "radius ");
```

```
fgets(buffer, INPUT_LINE_SIZE, infile);
if(valid_num_string(buffer)) {
  temp_op_ptr->op->color = atoi(buffer);
  if(temp_op_ptr->op->color > MAXCOLORS) {
    add_string(&error_str_ptr, "color ");
    temp_op_ptr->op->color = MAXCOLORS;
  }
}
else
  add_string(&error_str_ptr, "color ");

if(strcmp("TRUE\n",
          fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
  temp_op_ptr->op->is_deleted = TRUE;
else
  if(strcmp("FALSE\n", buffer) == 0)
    temp_op_ptr->op->is_deleted = FALSE;
  else
    add_string(&error_str_ptr, "is_deleted ");

if(strcmp("TRUE\n",
          fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
  temp_op_ptr->op->is_new = TRUE;
else
  if(strcmp("FALSE\n", buffer) == 0)
    temp_op_ptr->op->is_new = FALSE;
  else
    add_string(&error_str_ptr, "is_deleted ");

if(strcmp("TRUE\n",
          fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
  temp_op_ptr->op->is_modified = TRUE;
else
  if(strcmp("FALSE\n", buffer) == 0)
    temp_op_ptr->op->is_modified = FALSE;
  else
    add_string(&error_str_ptr, "is_deleted ");

temp_op_ptr->op->is_composite =
                              (BOOL *) malloc(sizeof(BOOL));
if(strcmp("TRUE\n",
          fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
  *(temp_op_ptr->op->is_composite) = TRUE;
else
  if(strcmp("FALSE\n", buffer) == 0)
    *(temp_op_ptr->op->is_composite) = FALSE;
  else
    add_string(&error_str_ptr, "is_deleted ");

if(strcmp("TRUE\n",
          fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
  temp_op_ptr->op->is_terminator = TRUE;
else
  if(strcmp("FALSE\n", buffer) == 0)
```

```
        temp_op_ptr->op->is_terminator = FALSE;
      else
        add_string(&error_str_ptr, "is_deleted ");


    if(strlen(error_str_ptr) > error_length) {
      printf("%s\n", error_str_ptr);
      status = FAILED;
    }
    free(error_str_ptr);

    if(ferror(infile)) {
      printf("Unix reports file errors building operator %s\n",
             temp_op_ptr->op->name);
      clearerr(infile);
      free(temp_op_ptr->op);
      free(temp_op_ptr);
      temp_op_ptr = NULL;
      status = FAILED;
    }
    *instatus = status;
    return temp_op_ptr;
  }
}

/*   Builds a spline node from disk.
*/

struct spline_node* build_sp_fm_disk(infile, instatus)
     FILE *infile;
     Status *instatus;
{
  char buffer[INPUT_LINE_SIZE + 1], *error_str_ptr = NULL;
  struct spline_node *head_ptr = NULL, *curr_ptr, *temp_sp_ptr;
  int error_length;
  BOOL done = FALSE;
  Status status = SUCCEEDED;

  error_str_ptr = strdup("Spline elements in error: ");
  error_length = strlen(error_str_ptr);

  fgets(buffer, INPUT_LINE_SIZE, infile); /* clear "SPLINE" */
  while(!done) {
    if((strcmp("SPLINEEND\n", fgets(buffer, INPUT_LINE_SIZE,
                                    infile)) != 0)
       && (!ferror(infile))) {
      temp_sp_ptr =
        (struct spline_node *)
          malloc(sizeof(struct spline_node));
      temp_sp_ptr->next = NULL;

      if(valid_num_string(buffer))
        temp_sp_ptr->x = atoi(buffer);
      else
```

```
        add_string(&error_str_ptr, "x  ");

      fgets(buffer, INPUT_LINE_SIZE, infile);
      if(valid_num_string(buffer))
        temp_sp_ptr->y = atoi(buffer);
      else
        add_string(&error_str_ptr, "y ");

      if(head_ptr == NULL) {
        head_ptr = temp_sp_ptr;
        curr_ptr = temp_sp_ptr;
      }
      else {
        curr_ptr->next = temp_sp_ptr;
        curr_ptr = curr_ptr->next;
      }
    }
    else
      done = TRUE;
  }

  if(strlen(error_str_ptr) > error_length) {
    printf("%s\n", error_str_ptr);
    status = FAILED;
  }
  free(error_str_ptr);

  if(ferror(infile)) {
    printf("Unix reports file errors building spline.\n");
    clearerr(infile);
    free(head_ptr);
    head_ptr = NULL;
    status = FAILED;
  }
  *instatus = status;
  return head_ptr;
}

/*   Returns a pointer to the operator node specified by the
  id #.
*/

struct op_node* find_op_node(id, head_op_ptr)
     OP_ID id;
     struct op_node *head_op_ptr;
{
  struct op_node *temp_op_ptr = head_op_ptr;

  while(temp_op_ptr != NULL)
    if(temp_op_ptr->op->id == id)
      return temp_op_ptr;
    else
      temp_op_ptr = temp_op_ptr->next;
  return NULL;
```

```
}

/*   Builds a stream node from disk.
*/

struct st_node* build_st_node_fm_disk(infile, head_op_ptr,
                                      instatus)
     FILE *infile;
     struct op_node* head_op_ptr;
     Status *instatus;
{
  struct st_node* temp_st_ptr =
    (struct st_node *) malloc(sizeof(struct st_node));
  char buffer[INPUT_LINE_SIZE + 1], *error_str_ptr = NULL;
  int last_char, temp_node_id, error_length;
  struct op_node* temp_opstr_ptr;
  Status status = SUCCEEDED;

  temp_st_ptr->st =
    (struct st_str *) malloc(sizeof(struct st_str));
  temp_st_ptr->next = NULL;
  fgets(buffer, INPUT_LINE_SIZE, infile);
  last_char = strlen(buffer) - 1;
  if(buffer[last_char] == '\n')
    buffer[last_char] = 0;
  if(strcmp(buffer, "ENDDATA") == 0) {
    free(temp_st_ptr->st);
    free(temp_st_ptr);
    return NULL;
  }
  else {
    temp_st_ptr->st->name = strdup(buffer);
    error_str_ptr = strdup(buffer);
    add_string(&error_str_ptr, " elements in error: ");
    error_length = strlen(error_str_ptr);

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer)) {
      temp_st_ptr->st->name_font = atoi(buffer);
      if(temp_st_ptr->st->name_font > MAXFONTS) {
        add_string(&error_str_ptr, "name_font ");
        temp_st_ptr->st->name_font = MAXFONTS;
      }
    }
    else
      add_string(&error_str_ptr, "name_font ");

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer))
      temp_st_ptr->st->name_x = atoi(buffer);
    else
      add_string(&error_str_ptr, "name_x ");

    fgets(buffer, INPUT_LINE_SIZE, infile);
```

```
    if(valid_num_string(buffer))
      temp_st_ptr->st->name_y = atoi(buffer);
    else
      add_string(&error_str_ptr, "name_y ");

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer))
      temp_st_ptr->st->id = atoi(buffer);
    else
      add_string(&error_str_ptr, "id ");

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer)) {
      temp_node_id = atoi(buffer);
      if(temp_node_id == 0)
        temp_st_ptr->st->from = NULL;
      else {
        temp_opstr_ptr =
          find_op_node(temp_node_id, head_op_ptr);
        if(temp_opstr_ptr == NULL) {
          printf("\"from\" operator node not found in build_st_node_fm_-
disk\n");
          add_string(&error_str_ptr, "from_id ");
        }
        temp_st_ptr->st->from = temp_opstr_ptr;
      }
    }
    else
      add_string(&error_str_ptr, "from_id ");

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer)) {
      temp_node_id = atoi(buffer);
      if(temp_node_id == 0)
        temp_st_ptr->st->to = NULL;
      else {
        temp_opstr_ptr =
          find_op_node(temp_node_id, head_op_ptr);
        if(temp_opstr_ptr == NULL) {
          printf("\"to\" operator node not found in build_st_node_fm_disk\n");
          add_string(&error_str_ptr, "to_id ");
        }
        temp_st_ptr->st->to = temp_opstr_ptr;
      }
    }
    else
      add_string(&error_str_ptr, "to_id ");

    temp_st_ptr->st->arc = build_sp_fm_disk(infile, &status);

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer))
      temp_st_ptr->st->latency = atoi(buffer);
    else
```

```
      add_string(&error_str_ptr, "latency ");

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer)) {
      temp_st_ptr->st->latency_font = atoi(buffer);
      if(temp_st_ptr->st->latency_font > MAXFONTS) {
        add_string(&error_str_ptr, "latency_font ");
        temp_st_ptr->st->latency_font = MAXFONTS;
      }
    }
    else
      add_string(&error_str_ptr, "latency_font ");

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer))
      temp_st_ptr->st->latency_x = atoi(buffer);
    else
      add_string(&error_str_ptr, "latency_x ");

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer))
      temp_st_ptr->st->latency_y = atoi(buffer);
    else
      add_string(&error_str_ptr, "latency_y ");

    if(strcmp("TRUE\n",
              fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
      temp_st_ptr->st->is_deleted = TRUE;
    else
      if(strcmp("FALSE\n", buffer) == 0)
        temp_st_ptr->st->is_deleted = FALSE;
      else
        add_string(&error_str_ptr, "is_deleted ");

    if(strcmp("TRUE\n",
              fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
      temp_st_ptr->st->is_new = TRUE;
    else
      if(strcmp("FALSE\n", buffer) == 0)
        temp_st_ptr->st->is_new = FALSE;
      else
        add_string(&error_str_ptr, "is_deleted ");

    if(strcmp("TRUE\n",
              fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
      temp_st_ptr->st->is_modified = TRUE;
    else
      if(strcmp("FALSE\n", buffer) == 0)
        temp_st_ptr->st->is_modified = FALSE;
      else
        add_string(&error_str_ptr, "is_deleted ");

    if(strcmp("TRUE\n",
              fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
```

```
      temp_st_ptr->st->is_state_variable = TRUE;
    else
      if(strcmp("FALSE\n", buffer) == 0)
        temp_st_ptr->st->is_state_variable = FALSE;
      else
        add_string(&error_str_ptr, "is_deleted ");

    if(strlen(error_str_ptr) > error_length) {
      printf("%s\n", error_str_ptr);
      status = FAILED;
    }

    free(error_str_ptr);

    if(ferror(infile)) {
      printf("Unix reports file errors building stream %s\n",
             temp_st_ptr->st->name);
      clearerr(infile);
      free(temp_st_ptr->st);
      free(temp_st_ptr);
      temp_st_ptr = NULL;
      status = FAILED;
    }
    *instatus = status;
    return temp_st_ptr;
  }
}

/*   Updates the syntax-directed editor's global data structure
  with the data read in from the disk file.  New objects are
  added to the linked list, while data from modified objects
  is copied over to the original object.

    First copies the data into all the existing objects, then
  goes back through to link in the new objects.
*/

void update_globals(head_op_ptr, head_st_ptr)
     struct op_node **head_op_ptr;
     struct st_node **head_st_ptr;
{
  struct op_node *temp_op_src_ptr = *head_op_ptr;
  struct op_node *temp_op_ptr,
    *temp_op_tgt_ptr = the_operator_list;
  struct st_node *temp_st_src_ptr = *head_st_ptr;
  struct st_node *temp_st_ptr,
    *temp_st_tgt_ptr = the_stream_list;
  struct spline_node *temp_spln_tgt_ptr, *temp_spln_src_ptr,
    *temp_spln_ptr;

#ifdef GE_DEBUG
printf("=============    Incoming      ====================\n");
printf("global\n");
  disp_ge_opnodes(the_operator_list);
```

```
  disp_ge_stnodes(the_stream_list);
printf("local\n");
  disp_ge_opnodes(*head_op_ptr);
  disp_ge_stnodes(*head_st_ptr);
#endif

  while(temp_op_src_ptr != NULL) {
    if(temp_op_src_ptr->op->is_new == FALSE) {
/* Update original node. */
      temp_op_tgt_ptr = the_operator_list;
      while(temp_op_tgt_ptr != NULL) {
        if(temp_op_tgt_ptr->op->id == temp_op_src_ptr->op->id) {

/*    Statement frees memory in the global data structure.  Left
    commented out to prevent possible SDE problems.  Not freeing
    this memory results in a memory leak.

          free(temp_op_tgt_ptr->op->name);
*/

          temp_op_tgt_ptr->op->name =
            strdup(temp_op_src_ptr->op->name);
          temp_op_tgt_ptr->op->name_font =
            temp_op_src_ptr->op->name_font;
          temp_op_tgt_ptr->op->name_x =
            temp_op_src_ptr->op->name_x;
          temp_op_tgt_ptr->op->name_y =
            temp_op_src_ptr->op->name_y;

          if(temp_op_src_ptr->op->met != NULL) {
            if(temp_op_tgt_ptr->op->met == NULL)
              temp_op_tgt_ptr->op->met =
                (int *) malloc(sizeof(int));
            *(temp_op_tgt_ptr->op->met) =
              *(temp_op_src_ptr->op->met);
          }
          else
            temp_op_tgt_ptr->op->met = NULL;
          temp_op_tgt_ptr->op->met_font =
            temp_op_src_ptr->op->met_font;
          temp_op_tgt_ptr->op->met_x =
            temp_op_src_ptr->op->met_x;
          temp_op_tgt_ptr->op->met_y =
            temp_op_src_ptr->op->met_y;
          temp_op_tgt_ptr->op->x = temp_op_src_ptr->op->x;
          temp_op_tgt_ptr->op->y = temp_op_src_ptr->op->y;
          temp_op_tgt_ptr->op->radius =
            temp_op_src_ptr->op->radius;
          temp_op_tgt_ptr->op->color =
            temp_op_src_ptr->op->color;

#ifdef GE_DEBUG
          printf("made it through the integers.\n");
#endif
```

```
          temp_op_tgt_ptr->op->is_deleted =
            temp_op_src_ptr->op->is_deleted;
          temp_op_tgt_ptr->op->is_new =
            temp_op_src_ptr->op->is_new;
          if(temp_op_tgt_ptr->op->is_composite == NULL)
            printf("NULL value for global \"is_composite\". Possible probl-
em.\n");
          else {
            *(temp_op_tgt_ptr->op->is_composite) =
              *(temp_op_src_ptr->op->is_composite);
          }
          temp_op_tgt_ptr->op->is_terminator =
            temp_op_src_ptr->op->is_terminator;
          temp_op_tgt_ptr->op->is_modified =
            temp_op_src_ptr->op->is_modified;
          break;
        }
        else
          temp_op_tgt_ptr = temp_op_tgt_ptr->next;
      }
    }
    temp_op_src_ptr = temp_op_src_ptr->next;
  }

#ifdef GE_DEBUG
  printf("finished operators\n");
#endif

  while(temp_st_src_ptr != NULL) {
    if(temp_st_src_ptr->st->is_new == FALSE) {
      temp_st_tgt_ptr = the_stream_list;
      while(temp_st_tgt_ptr != NULL) {
        if(temp_st_tgt_ptr->st->id == temp_st_src_ptr->st->id) {

/*    Statement frees memory in the global data structure.  Left
    commented out to prevent possible SDE problems.  Not freeing
    this memory results in a memory leak.

          free(temp_st_tgt_ptr->st->name);
*/

          temp_st_tgt_ptr->st->name =
            strdup(temp_st_src_ptr->st->name);
          temp_st_tgt_ptr->st->name_font =
            temp_st_src_ptr->st->name_font;
          temp_st_tgt_ptr->st->name_x =
            temp_st_src_ptr->st->name_x;
          temp_st_tgt_ptr->st->name_y =
            temp_st_src_ptr->st->name_y;
          if(temp_st_src_ptr->st->from != NULL) {
            temp_op_ptr =
              find_op_node(temp_st_src_ptr->st->from->op->id,
                           the_operator_list);
```

```
            if(temp_op_ptr == NULL)
             printf("\"from\" operator not found in global data structure\n");
            temp_st_tgt_ptr->st->from = temp_op_ptr;
          }
          else
            temp_st_tgt_ptr->st->from = NULL;
          if(temp_st_src_ptr->st->to != NULL) {
              temp_op_ptr =
                find_op_node(temp_st_src_ptr->st->to->op->id,
                             the_operator_list);
              if(temp_op_ptr == NULL)
               printf("\"to\" operator not found in global data structure\n");
              temp_st_tgt_ptr->st->to = temp_op_ptr;
          }
          else
            temp_st_tgt_ptr->st->to = NULL;

#ifdef GE_DEBUG
          printf("Made it to the spline.\n");
#endif

          if(temp_st_src_ptr->st->arc != NULL) {

/*    Statements free memory in the global data structure.  Left
    commented out to prevent possible SDE problems.  Not freeing
    this memory results in a memory leak.

            if(temp_st_tgt_ptr->st->arc != NULL) {
              temp_spln_tgt_ptr = temp_st_tgt_ptr->st->arc;
              while(temp_spln_tgt_ptr != NULL) {
                temp_spln_ptr = temp_spln_tgt_ptr->next;
                free(temp_spln_tgt_ptr);
                temp_spln_tgt_ptr = temp_spln_ptr;
              }
            }

#ifdef GE_DEBUG
            printf("Freed the existing spline\n");
#endif

*/

            temp_spln_src_ptr = temp_st_src_ptr->st->arc;
            temp_st_tgt_ptr->st->arc  =
              (struct spline_node *)
                malloc(sizeof(struct spline_node));
            temp_st_tgt_ptr->st->arc->x = temp_spln_src_ptr->x;
            temp_st_tgt_ptr->st->arc->y = temp_spln_src_ptr->y;
            temp_spln_src_ptr = temp_spln_src_ptr->next;
            temp_spln_tgt_ptr = temp_st_tgt_ptr->st->arc;

#ifdef GE_DEBUG
            printf("Set head\n");
#endif
```

```
            while(temp_spln_src_ptr != NULL) {
              temp_spln_tgt_ptr->next =
                (struct spline_node *)
                  malloc(sizeof(struct spline_node));
              temp_spln_tgt_ptr = temp_spln_tgt_ptr->next;

              temp_spln_tgt_ptr->x = temp_spln_src_ptr->x;
              temp_spln_tgt_ptr->y = temp_spln_src_ptr->y;
              temp_spln_src_ptr = temp_spln_src_ptr->next;
            }
            temp_spln_tgt_ptr->next = NULL;
          }
          else
            temp_st_tgt_ptr->st->arc = NULL;

#ifdef GE_DEBUG
          printf("made it past the spline.\n");
#endif

          temp_st_tgt_ptr->st->latency =
            temp_st_src_ptr->st->latency;
          temp_st_tgt_ptr->st->latency_font =
            temp_st_src_ptr->st->latency_font;
          temp_st_tgt_ptr->st->latency_x =
            temp_st_src_ptr->st->latency_x;
          temp_st_tgt_ptr->st->latency_y =
            temp_st_src_ptr->st->latency_y;
          temp_st_tgt_ptr->st->is_deleted =
            temp_st_src_ptr->st->is_deleted;
          temp_st_tgt_ptr->st->is_new =
            temp_st_src_ptr->st->is_new;
          temp_st_tgt_ptr->st->is_state_variable =
            temp_st_src_ptr->st->is_state_variable;
          temp_st_tgt_ptr->st->is_modified =
            temp_st_src_ptr->st->is_modified;
          break;
        }
        else
          temp_st_tgt_ptr = temp_st_tgt_ptr->next;
      }
    }
    temp_st_src_ptr = temp_st_src_ptr->next;
  }

#ifdef GE_DEBUG
  printf("finished streams\n");
#endif

/*   Links in the new objects.
*/

  while(*head_op_ptr != NULL) {
    temp_op_src_ptr = *head_op_ptr;
```

```
    if((*head_op_ptr)->op->is_new == TRUE) {
      *head_op_ptr = (*head_op_ptr)->next;
      temp_op_src_ptr->next = the_operator_list;
      the_operator_list = temp_op_src_ptr;
    }
    else
      break;
  }
  if(*head_op_ptr != NULL) {
    temp_op_src_ptr = *head_op_ptr;
    while(temp_op_src_ptr->next != NULL) {
      if(temp_op_src_ptr->next->op->is_new == TRUE) {
        temp_op_ptr = temp_op_src_ptr->next;
        temp_op_src_ptr->next = temp_op_src_ptr->next->next;
        temp_op_ptr->next = the_operator_list;
        the_operator_list = temp_op_ptr;
      }
      else
        temp_op_src_ptr = temp_op_src_ptr->next;
    }
  }
  while(*head_st_ptr != NULL) {
    temp_st_src_ptr = *head_st_ptr;
    if((*head_st_ptr)->st->is_new == TRUE) {
      *head_st_ptr = (*head_st_ptr)->next;
      temp_st_src_ptr->next = the_stream_list;
      the_stream_list = temp_st_src_ptr;
    }
    else
      break;
  }
  if(*head_st_ptr != NULL) {
    temp_st_src_ptr = *head_st_ptr;
    while(temp_st_src_ptr->next != NULL) {
      if(temp_st_src_ptr->next->st->is_new == TRUE) {
        temp_st_ptr = temp_st_src_ptr->next;
        temp_st_src_ptr->next = temp_st_src_ptr->next->next;
        temp_st_ptr->next = the_stream_list;
        the_stream_list = temp_st_ptr;
      }
      else
        temp_st_src_ptr = temp_st_src_ptr->next;
    }
  }

#ifdef GE_DEBUG
printf("==============    Outgoing      ========================\n");
printf("global\n");
  disp_ge_opnodes(the_operator_list);
  disp_ge_stnodes(the_stream_list);
printf("local\n");
  disp_ge_opnodes(*head_op_ptr);
  disp_ge_stnodes(*head_st_ptr);
#endif
```

```
/*   If new objects have been added to the beginning of the
  list, the syntax-directed editor's pointers to the head
  of list must be modified to point to the new head of the
  list.
*/

  current_graph->operator_list = the_operator_list;
  current_graph->stream_list = the_stream_list;
}

/*   Frees the memory allocated to the given lists. */

/* Should be left commented out until the SDE is more stable.
void free_data(head_op_ptr, head_st_ptr)
     struct op_node *head_op_ptr;
     struct st_node *head_st_ptr;
 {
  struct op_node* op_ptr_table[MAX_OPERATORS];
  struct op_node* temp_op_ptr = head_op_ptr;
  struct st_node* st_ptr_table[MAX_STREAMS];
  struct st_node* temp_st_ptr = head_st_ptr;
  struct spline_node* spline_ptr_table[MAX_SPLINE_NODES];
  struct spline_node* temp_node_ptr;
  int index = 0, spline_index = 0, i, j;

  while(temp_op_ptr != NULL) {
    op_ptr_table[index] = temp_op_ptr;
    temp_op_ptr = temp_op_ptr->next;
    index++;
  }

  for(i = 0; i < index; i++) {
    free(op_ptr_table[i]->op->name);
    free(op_ptr_table[i]->op->met);
    free(op_ptr_table[i]->op->is_composite);
    free(op_ptr_table[i]->op);
    free(op_ptr_table[i]);
  }

  index = 0;
  while(temp_st_ptr != NULL) {
    st_ptr_table[index] = temp_st_ptr;
    temp_st_ptr = temp_st_ptr->next;
    index++;
  }

  for(i = 0; i < index; i++) {
    spline_index = 0;
    temp_node_ptr = st_ptr_table[i]->st->arc;
    while(temp_node_ptr != NULL) {
      spline_ptr_table[spline_index] = temp_node_ptr;
      temp_node_ptr = temp_node_ptr->next;
      spline_index++;
```

```
    }
    for(j = 0; j < spline_index; j++)
      free(spline_ptr_table[j]);
    free(st_ptr_table[i]->st->name);
    free(st_ptr_table[i]->st);
    free(st_ptr_table[i]);
  }
}

*/

/*   Retrieves the updated graph data from disk. */

Status retrieve_data() {
  FILE *infile;
  char buffer[INPUT_LINE_SIZE + 1], *error_str_ptr = NULL;
  struct op_node *temp_op_ptr, *head_op_ptr;
  struct st_node *temp_st_ptr, *head_st_ptr;
  int return_info, temp_int, last_char;
  Status status = SUCCEEDED;

  infile = fopen("gedatatransfile2.txt", "rt");
  if(infile != NULL) {
    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(strcmp(buffer, "GE93DATAFILE\n") == 0) {
      fgets(buffer, INPUT_LINE_SIZE, infile);
      last_char = strlen(buffer) - 1;
      if(buffer[last_char] == '\n')
        buffer[last_char] = 0;
      current_graph->name = strdup(buffer);
      fgets(buffer, INPUT_LINE_SIZE, infile);
      if(valid_num_string(buffer)) {
        current_graph->name_font = atoi(buffer);
        if(current_graph->name_font > MAXFONTS)
          error_str_ptr = strdup("Name Font Too Large");
      }
      else
        strdup(error_str_ptr, "Corrupted header information");

      if(error_str_ptr == NULL) {
        fgets(buffer, INPUT_LINE_SIZE, infile);
        if(valid_num_string(buffer))
          current_graph->name_x = atoi(buffer);
        else
          error_str_ptr = strdup("Corrupted name_x");
      }

      if(error_str_ptr == NULL) {
        fgets(buffer, INPUT_LINE_SIZE, infile);
        if(valid_num_string(buffer))
          current_graph->name_y = atoi(buffer);
        else
          error_str_ptr = strdup("Corrupted name_y");
      }
```

```
      if(error_str_ptr != NULL)
        status = FAILED;
      if(status == SUCCEEDED) {  /* #1 */
           /* Clears OPERATORS keyword in file. */
        fgets(buffer, INPUT_LINE_SIZE, infile);

#ifdef GE_DEBUG
printf("reading\n");
#endif
        head_op_ptr = build_op_node_fm_disk(infile, &status);

#ifdef GE_DEBUG
disp_ge_opnodes(head_op_ptr);
#endif

        if((head_op_ptr != NULL) && (status == SUCCEEDED)) {
          temp_op_ptr = head_op_ptr;
          while((temp_op_ptr != NULL) && (status == SUCCEEDED)) {
            temp_op_ptr->next =
              build_op_node_fm_disk(infile, &status);
            temp_op_ptr = temp_op_ptr->next;

#ifdef GE_DEBUG
disp_ge_opnodes(temp_op_ptr);
#endif

          }
        }

        if(status == SUCCEEDED) {
          head_st_ptr =
            build_st_node_fm_disk(infile, head_op_ptr, &status);

#ifdef GE_DEBUG
disp_ge_stnodes(head_st_ptr);
#endif

          if((head_st_ptr != NULL) && (status == SUCCEEDED)) {
            temp_st_ptr = head_st_ptr;
            while((temp_st_ptr != NULL) &&
                  (status == SUCCEEDED)) {
              temp_st_ptr->next =
                build_st_node_fm_disk(infile,
                                      head_op_ptr, &status);
              temp_st_ptr = temp_st_ptr->next;

#ifdef GE_DEBUG
disp_ge_stnodes(temp_st_ptr);
#endif

            }
          }
        }
```

```
        if(status == SUCCEEDED) { /* #2  */
          return_info = atoi(fgets(buffer, INPUT_LINE_SIZE,
                                   infile));
          if(return_info == ERROR)
            printf("Graphic editor returned an error.\n");
          else {

#ifdef GE_DEBUG
            printf("Retrieve data ============================\nglobal:\n");
            disp_ge_opnodes(the_operator_list);
            disp_ge_stnodes(the_stream_list);
            printf("local:\n");
            disp_ge_opnodes(head_op_ptr);
            disp_ge_stnodes(head_st_ptr);
#endif

            if(return_info != NOUPDATE) {
              update_globals(&head_op_ptr, &head_st_ptr);
              if(return_info == GRUP) /* change UP */
                level_change_direction = UP;
              else
                if(return_info == SAME)
                  level_change_direction = SAME;
                else {
                  level_change_direction = DOWN;
                  goto_child = strdup(fgets(buffer,
                                            INPUT_LINE_SIZE,
                                            infile));
                }
            }
            else
              level_change_direction = SAME;
          }
        } /* #2  */
/*      free_data(head_op_ptr, head_st_ptr); */
      } /* #1 */
    }
    else {
      printf("Data file format is questionable.\n");
      status = FAILED;
    }
    fclose(infile);
  }
  else {
    printf("Input data tranfer file not found.\n");
    status = FAILED;
  }
  if(error_str_ptr != NULL) {
    printf(error_str_ptr);
    free(error_str_ptr);
  }
  return status;
}
```

```
/*   Displays the type of an event.  Debug code left in for
  future use.
*/

static int id_event(event_to_check)
XEvent event_to_check;
{
  switch(event_to_check.type) {
  case KeyPress:
    printf("KeyPress");
    break;
  case KeyRelease:
    printf("KeyRelease");
    break;
  case ButtonPress:
    printf("ButtonPress");
    break;
  case ButtonRelease:
    printf("ButtonRelease");
    break;
  case MotionNotify:
    printf("MotionNotify");
    break;
  case EnterNotify:
    printf("EnterNotify");
    break;
  case LeaveNotify:
    printf("LeaveNotify");
    break;
  case FocusIn:
    printf("FocusIn");
    break;
  case FocusOut:
    printf("FocusOut");
    break;
  case KeymapNotify:
    printf("KeymapNotify");
    break;
  case Expose:
    printf("Expose");
    break;
  case GraphicsExpose:
    printf("GraphicsExpose");
    break;
  case NoExpose:
    printf("NoExpose");
    break;
  case VisibilityNotify:
    printf("VisibilityNotify");
    break;
  case CreateNotify:
    printf("CreateNotify");
    break;
```

```
case DestroyNotify:
  printf("DestroyNotify");
  break;
case UnmapNotify:
  printf("UnmapNotify");
  break;
case MapNotify:
  printf("MapNotify");
  break;
case MapRequest:
  printf("MapRequest");
  break;
case ReparentNotify:
  printf("ReparentNotify");
  break;
case ConfigureNotify:
  printf("ConfigureNotify");
  break;
case ConfigureRequest:
  printf("ConfigureRequest");
  break;
case GravityNotify:
  printf("GravityNotify");
  break;
case ResizeRequest:
  printf("ResizeRequest");
  break;
case CirculateNotify:
  printf("CirculateNotify");
  break;
case CirculateRequest:
  printf("CirculateRequest");
  break;
case PropertyNotify:
  printf("PropertyNotify");
  break;
case SelectionClear:
  printf("SelectionClear");
  break;
case SelectionRequest:
  printf("SelectionRequest");
  break;
case SelectionNotify:
  printf("SelectionNotify");
  break;
case ColormapNotify:
  printf("ColormapNotify");
  break;
case ClientMessage:
  printf("ClientMessage");
  break;
case MappingNotify:
  printf("MappingNotify");
  break;
```

```
  default:
    printf(event_to_check.type);
  }
  printf(" Event, send_event ");
  if(event_to_check.xany.send_event)
    printf("TRUE");
  else
    printf("FALSE");
  printf(", Target Window: %x\n",  event_to_check.xany.window);
  return event_to_check.type;
}

/*   Writes out data from the header node.
*/

void write_header_data(outfile)
FILE *outfile;
{
  char buffer[INPUT_LINE_SIZE + 1];

  if(current_graph->name == NULL)
    fprintf(outfile, "\n");
  else
    fprintf(outfile, "%s\n", current_graph->name);
  sprintf(buffer, "%d\n", current_graph->name_font);
  fprintf(outfile, buffer);
  sprintf(buffer, "%d\n", current_graph->name_x);
  fprintf(outfile, buffer);
  sprintf(buffer, "%d\n", current_graph->name_y);
  fprintf(outfile, buffer);
}

/*   Writes objects to disk.
*/

STATUS store_data() {
  FILE *outfile;
  struct op_node* temp_op_node;
  struct st_node* temp_st_node;
  STATUS status = SUCCEEDED;

#ifdef GE_DEBUG
printf("store data in ============================\nglobal:\n");
      disp_ge_opnodes(the_operator_list);
      disp_ge_stnodes(the_stream_list);
      printf("Setting the lists\n");
#endif

  if(current_graph != NULL) {
    the_operator_list = current_graph->operator_list;
    the_stream_list = current_graph->stream_list;

#ifdef GE_DEBUG
printf("store data out============================\nglobal:\n");
```

```
      disp_ge_opnodes(the_operator_list);
      disp_ge_stnodes(the_stream_list);
#endif

    outfile = fopen("gedatatransfile.txt", "wt");
    if(outfile != NULL) {
      fprintf(outfile, "GE93DATAFILE\n");
      write_header_data(outfile);
      fprintf(outfile, "OPERATORS\n");
      temp_op_node = the_operator_list;
      while(temp_op_node != NULL) {
        status = write_op_to_file(outfile, temp_op_node);
        if(status == SUCCEEDED)
          temp_op_node = temp_op_node->next;
        else
          break;
      }
      if(status == SUCCEEDED) {
        temp_st_node = the_stream_list;
        fprintf(outfile, "STREAMS\n");
        while(temp_st_node != NULL) {
          status = write_st_to_file(outfile, temp_st_node);
          if(status == SUCCEEDED)
            temp_st_node = temp_st_node->next;
          else
            break;
        }
      }
      fprintf(outfile, "ENDDATA\n");
      fclose(outfile);
      if(status == FAILED)
        printf("There were problems writing the output file.\n");
    }
    else {
    printf("Unable to open graph editor data transfer file.\n");
      clearerr(outfile);
      status = FAILED;
    }
  }
  else {
    printf("current_graph pointer is NULL. Unable to display graph.\n");
    status = FAILED;
  }
  return status;
}

/*   Sends an X ClientMessage event to the graph viewer so it
  knows what it's supposed to do.
*/

void send_event(message)
     char *message;
{
  Status status;
```

```
  int argc = 1;
  char **argv = (char **) malloc(2 * sizeof(char *));

  static Display *display_ptr = NULL;
  static Window window;
  static Widget toplevel, label;
  static XtAppContext app;
  static char *data[2];
  static XTextProperty text_prop_return;
  static Atom property_name, display_id_atom, actual_type_atom;
  static int actual_format;
  static unsigned long items_returned, bytes_remaining;
  static Window *property_data;
  static unsigned char *property_pointer;
    XmString     xmstr;
    Arg          args[10];
    int          n;

  XEvent *big_event = (XEvent *) malloc(sizeof(XEvent));
  XEvent in_event;

/*   If the display ptr hasn't been initialized, this is
  the first time through the routine.  Get the root window
  id and retrieve the graph viewer's window id from the
  X property area.
*/

  if(display_ptr == NULL) {
    argv[0] = strdup("sde");
    argv[1] = NULL;

/*   This is done to get the root window of the display.
  Although no attempt is made to change anything, this is
  called from within a running X program, the syntax-directed
  editor, and X sometimes complains about the same application
  initializing twice.
*/
    toplevel = XtVaAppInitialize(&app, "dummy_app", NULL, 0,
                                 &argc, argv, NULL, NULL);
    display_ptr = XtDisplay(toplevel);
    window = DefaultRootWindow(display_ptr);
    data[0] = strdup("REFRESH");
    data[1] = NULL;
    XStringListToTextProperty(data, 1, &text_prop_return);
    display_id_atom = XInternAtom(display_ptr, "WINDOW_ID",
                                  False);
    XGetWindowProperty(display_ptr, window, display_id_atom, 0,
                       1, False, XA_WINDOW, &actual_type_atom,
                       &actual_format, &items_returned,
                       &bytes_remaining, &property_pointer);
    property_data = (Window *) property_pointer;
  }

  big_event->type = ClientMessage;
```

```
  big_event->xclient.window = *property_data;
  big_event->xclient.format = 8;
  strcpy(big_event->xclient.data.b, message);
  status = XSendEvent(display_ptr, *property_data, True, 0,
                      big_event);
  XFlush(display_ptr);
}

/*   Writes data to disk and tells the graph viewer to refresh
  its display from disk.
*/

void refresh() {
  STATUS status;

  status = store_data();
  if(status == SUCCEEDED)
    send_event("GEDATAIN");
  else
    printf("Problems writing data for graph viewer.\n");
}

/*   Invokes the graph editor.
*/

void edit() {
  STATUS status;
  FILE *tool_file;
  char buffer[INPUT_LINE_SIZE + 1],
    filebuffer[2 * INPUT_LINE_SIZE], home_dir;
  struct passwd *user_pass;

  status = store_data();
  if(status == SUCCEEDED) {
    user_pass = getpwuid(getuid());
    strcpy(filebuffer, user_pass->pw_dir);
    strcat(filebuffer, "/bin/tool_location.txt");
    if((tool_file = fopen(filebuffer, "rt")) == NULL)
      printf("Tool location file not found.\n");
    else {
      fscanf(tool_file, "%s", buffer);
      while((strcmp(buffer, "graph_editor:") != 0) &&
            (!feof(tool_file)))
        fscanf(tool_file, "%s", buffer);
      if(!feof(tool_file)) {
        fgets(buffer, INPUT_LINE_SIZE, tool_file);
        system(buffer);
        status = retrieve_data();
        if(status == SUCCEEDED) {

#ifdef GE_DEBUG
          if(level_change_direction == UP)
            printf("Change up\n");
          else
```

```
            if(level_change_direction == SAME)
              printf("No level change.\n");
            else
              printf("Change level: %s\n", goto_child);
#endif
        }
        else
          printf("Unable to correctly interpret edited data.\n");
      }
      else
        printf("Tool file did not contain the \"graph_editor\" parameter.\n");
    }
  }
  else
    printf("Problems storing data for graph editor\n");
  fclose(tool_file);
}

void kill_viewer() {

  send_event("GEQUIT");
}
```

```
/* ************************************************************

Name:           globals.h
Author:         Capt Robert M. Dixon
Program:        sde
Date Modified: 12 Sep 92
Remarks:   Provides the global variable declarations for
          the syntax-directed editor and the interface
          routines.

            Originally created by Professor Valdis Berzins and
          modified by Professor Fernando Naveda and Capt Robert
          M. Dixon.


************************************************************* */
#ifndef GLOBALS_H
#define GLOBALS_H 1

OPNodePTR the_operator_list;
ST_PTR the_stream_list;
HeadPtr prototype;
HeadPtr current_graph;
int op_id_count;
int level_change_direction;
char *goto_child;

#endif
```

```
/* ************************************************************

Name:           graph_editor.C
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 21 Sep 92
Remarks:    graph_editor.C is the main program for the
          CAPS '93 graph editor.- Depending on the command line
          parameters, it allows either viewing only or full
          editing of a graph passed by the CAPS '93 syntax-
          directed editor.

            graph_editor accepts one command line
          parameter.  Invoked with the -v parameter, it runs in
          view mode, allowing only viewing of the graph.
          Otherwise, it allows full screen editing.

            When invoked, graph_editor looks for an input
          text file specifying the attributes of the graph.
          This file is currently named 'gedatatransfile.txt'.
          If this file is not found the editor uses a blank
          canvas.

            Unless specified otherwise, on termination
          the editor writes the attributes of the current
          graph in an output text file currently named
          'gedatatransfile2.txt'.

General Comments:

            The XmProcessTraversal function is called numerous
          places in an attempt to keep the keyboard input focus
          in the drawing window.  This allows the editor to
          respond to the delete and backspace key.  This works
          with varying degrees of success.

Credits:    Portions of code are adapted from the following:

          Barakati, Naba, X Window System Programming, SAMS,
            1991.

          Heller, Dan, Motif Programming Manual, O'Reilly and
            Associates, 1991.

          Johnson, Eric, and Reichard, Kevin, X Window
            Applications Programming, MIS Press, 1989.

          Young, Douglas, Object Oriented Programming With C++
            and OSF/Motif, Prentice-Hall, 1992.

************************************************************ */

#include <stdlib.h>
#include <Xm/MainW.h>
```

```
#include <Xm/DrawingA.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/DrawnB.h>
#include <Xm/Form.h>
#include <Xm/TextF.h>
#include <Xm/Text.h>
#include <stream.h>
#include <Xm/SelectioB.h>
#include <Xm/LabelG.h>
#include <Xm/ToggleBG.h>
#include <X11/Xatom.h>
#include <Xm/List.h>
#include <Xm/MessageB.h>
#include "graph_object_list.h"
#include "operator_object.h"
#include "spline_object.h"
#include "stream_object.h"
#include "ge_defs.h"

//   MAXCOLORS is the number of colors defined to the editor.
// To add or subtract colors, this value must be modified.

#define BUTTONWIDTH 75

//   graph_editor has a number of global variables due to
// Motif's use of callback functions. Since these functions
// have fixed formal parameter lists, global variables must
// be used to pass some data between functions.

//   All drawing commands are executed on both drawing_a and
// drawing_area_pixmap. drawing_a is the visible canvas, while
// drawing_area_pixmap provides a backup.  When the canvas needs
// to be redrawn, the drawing in drawing_area pixmap is merely
// copied back onto the canvas.

//   colors[] is a list of predefined X colors. To use others,
// consult an X reference giving allowable color names. Using
// the predefined colors allows the user to specify color
// preferences in X resource text files.

//   graphic_list is a GraphObjectList containing all the
// visible operators and streams.

//   selected_object_ptr always points to the object selected
// (i. e. with handles around it) on the drawing canvas.

//   num_del_ops is the number of deleted operators, and
// del_op_id is an array of identifiers for deleted operators.

//   The Resrcs struct, resources[], and options[] are used
// by Motif for parsing the command line options.

GC std_graphics_context, dotted_context, erase_context;
```

```
Dimension  width, height;
Pixmap drawing_area_pixmap;
Widget drawing_a, graph_title, tool_indicator;
BOOLEAN state_stream = FALSE, alt_selected = FALSE;
char* colors[] = {"Aquamarine", "Black", "Blue", "BlueViolet",
                  "Brown", "CadetBlue", "Coral",
                  "CornflowerBlue", "Cyan", "DarkGreen",
                  "DarkOliveGreen", "DarkOrchid",
                  "DarkSlateBlue", "DarkSlateGrey",
                  "DarkTurquoise", "DimGrey", "Firebrick",
                  "ForestGreen", "Gold", "Goldenrod", "Grey",
                  "Green", "GreenYellow", "IndianRed", "Khaki",
                  "LightBlue", "LightGrey", "LightSteelBlue",
                  "LimeGreen", "Magenta", "Maroon",
                  "MediumAquamarine", "MediumBlue",
                  "MediumOrchid", "MediumSeaGreen",
                  "MediumSlateBlue", "MediumSpringGreen",
                  "MediumTurquoise", "MediumVioletRed",
                  "MidnightBlue", "Navy", "Orange", "OrangeRed",
                  "Orchid", "PaleGreen", "Pink", "Plum", "Red",
                  "Salmon", "SeaGreen", "Sienna", "SkyBlue",
                  "SlateBlue", "SpringGreen", "SteelBlue",
                  "Tan", "Thistle", "Turquoise", "Violet",
                  "VioletRed", "Wheat", "White", "Yellow",
                  "YellowGreen"};
unsigned long color_table[MAXCOLORS + 1];
TOOL_STATE tool_state = SELECT_TOOL;
GraphObjectList graphic_list;
GraphObject* selected_object_ptr = NULL;
Display *display_ptr;
Window draw_window;
int default_color = WHITE;
int default_font = COURIERBOLD12;
int num_del_ops = 0, del_op_id[MAXDELETEDOPS];

struct _resrcs {
  int viewer;
} Resrcs;

static XtResource resources[] = {
  {"viewer", "Viewer", XmRBoolean, sizeof (int),
   XtOffsetOf(struct _resrcs,viewer), XmRImmediate, False},
};

static XrmOptionDescRec options[] = {
  {"-v", "viewer", XrmoptionNoArg, "True"},
};

//   error_box displays a Motif dialog box with the given
// error message.  The dialog box is created the first time
// it is used.  Subsequent calls change the text string and
// redisplay the dialog box.

void error_box(char *error_message) {
```

```
  static Widget error_dialog = NULL;
  Arg arg[1];
  XmString t;

  if(error_dialog == NULL) {
    error_dialog = XmCreateMessageDialog(drawing_a, "error",
                     arg, 0);
    XtVaSetValues(XtParent(error_dialog),
                  XtNtitle, "Error",
                  NULL);
    XtUnmanageChild(XmMessageBoxGetChild(error_dialog,
                    XmDIALOG_HELP_BUTTON)); //  Hide help button
  }
  t = XmStringCreateSimple(error_message);
  XtVaSetValues(error_dialog,
                XmNmessageString, t,
                NULL);
  XmStringFree(t);
  XtManageChild(error_dialog);
}

//   Initializes the color table.

void initialize_color_table(Screen *screen) {
  Colormap color_map = DefaultColormapOfScreen(screen);
  XColor color, unused;
  int i, screen_depth = DefaultDepthOfScreen(screen);

  if(screen_depth > 1) {           //   a color screen
    for(i = 1; i <= MAXCOLORS; i++) {
      if(!XAllocNamedColor(display_ptr, color_map,
         colors[i - 1], &color, &unused))
           cout << "Allocated unknown color: " << colors[i - 1]
             << endl;
      color_table[i] = color.pixel;
    }
  }
  else {                           //   a black and white screen
    for(i = 1; i <= MAXCOLORS; i++) {
      if(strcmp(colors[i - 1], "White") != 0)
        color_table[i] = BlackPixelOfScreen(screen);
      else
        color_table[i] = WhitePixelOfScreen(screen);
    }
  }
}

//   Executes menu options from the 'graph' menu. This is
// called by either the menu callback function, if the
// pulldown menus are used, or by the draw() function,
// if the alt-key combinations are used.

void handle_graph_options(int item_no) {
  char buffer[INPUT_LINE_SIZE];
```

```
  XFlush(display_ptr);
  switch(item_no) {
  case 0:
    graphic_list.write_to_disk("gedatatransfile.txt",
                               SAME);
    break;
  case 1:
    graphic_list.build_from_disk();
    graphic_list.draw();
    break;
  case 2:
    sprintf(buffer, "xwd -id %d | xpr -device ps | lpr -hPcap",
            XtWindow (drawing_a));
    system (buffer);
    break;
  case 3:
    if(selected_object_ptr == NULL)
      error_box("Please select an operator");
    else {
      if(selected_object_ptr->is_a() == OPERATOROBJECT) {
        graphic_list.write_to_disk("gedatatransfile2.txt",
                                   selected_object_ptr->id());
        exit(0);
      }
      else
        error_box("Please select an operator");
    }
    break;
  case 4:
    graphic_list.write_to_disk("gedatatransfile2.txt", UP);
    exit(0);
  case 5:
    graphic_list.write_to_disk("gedatatransfile2.txt", SAME);
    exit(0);
    break;
  case 6:
    graphic_list.write_to_disk("gedatatransfile2.txt", NOUPDATE);
    exit(0);
    break;
  default:
    break;
  }
}

//   This function is called by the window manager when a
// menu option from the 'graph' pulldown menu is selected.  Its
// address is passed as a parameter to X when the menus are
// defined.

static void graph_cb(Widget, XtPointer client_data, XtPointer) {
  int item_no = (int) client_data;

  handle_graph_options(item_no);
```

```
}

//   This function is called when a selection is made from
// the list box displayed in the 'draw_options:Color' menu.

void color_list_cb(Widget widget, XtPointer,
                   XtPointer cb_struct_ptr) {
  XmListCallbackStruct *list_struct_ptr =
    (XmListCallbackStruct *) cb_struct_ptr;

  if(selected_object_ptr != NULL) {
    if(selected_object_ptr->is_a() == OPERATOROBJECT) {
      selected_object_ptr->erase();
      selected_object_ptr->set_color(
                                list_struct_ptr->item_position);
      selected_object_ptr->draw(SOLID);
    }
  }
  else
    default_color = list_struct_ptr->item_position;
  XtDestroyWidget(widget);
}

//   This function is called when a selection is made from
// the list box displayed in the 'draw_options:Font' menu.

void font_list_cb(Widget widget, XtPointer,
                  XtPointer cb_struct_ptr) {
  XmListCallbackStruct *list_struct_ptr =
    (XmListCallbackStruct *) cb_struct_ptr;

  if(selected_object_ptr != NULL) {
    selected_object_ptr->erase();
    selected_object_ptr->set_object_font(
                                list_struct_ptr->item_position);
    selected_object_ptr->draw(SOLID);
  }
  else {
    default_font = list_struct_ptr->item_position;
    graphic_list.set_default_font(default_font);
  }
  XtDestroyWidget(widget);
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
}

//   This function is called when a selection is made from
// the list box displayed in the 'draw_options:Undelete Operator'
// menu.

static void op_list_cb(Widget widget, XtPointer,
                       XtPointer cb_struct_ptr) {
  XmListCallbackStruct *list_struct_ptr =
    (XmListCallbackStruct *) cb_struct_ptr;
```

```
//   The last entry in the list is 'Cancel'.
  if(list_struct_ptr->item_position != num_del_ops + 1) {
    graphic_list.set_undeleted(OPERATOROBJECT,
                del_op_id[list_struct_ptr->item_position - 1]);
    graphic_list.draw();
  }
  XtDestroyWidget(widget);
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
}

//   Executes menu options from the 'Draw Options' menu. This is
// called by either the menu callback function, if the
// pulldown menus are used, or by the draw() function,
// if the alt-key combinations are used.

void handle_draw_options(int item_no) {
  int i, num_items = XtNumber(colors);
  XmStringTable color_list, font_list, op_list;
  Widget list_box, op_box;
  char *del_op_str[MAXDELETEDOPS];

  switch(item_no) {
    case 0:
      color_list =
        (XmStringTable) XtMalloc(num_items * sizeof(XmString *));
      for(i = 0; i < num_items; i++)
        color_list[i] = XmStringCreateSimple(colors[i]);
      list_box =
             XmCreateScrolledList(drawing_a, "Colors", NULL, 0);
      XtVaSetValues(list_box,
        XmNitems, color_list,
        XmNitemCount, num_items,
        XmNvisibleItemCount, 8,
        NULL);
      for(i = 0; i < num_items; i++)
        XmStringFree(color_list[i]);
      XtFree((char *) color_list);
      XtAddCallback(list_box, XmNdefaultActionCallback,
                    color_list_cb, NULL);
      XtManageChild(list_box);
      break;
    case 1:
      font_list =
         (XmStringTable) XtMalloc(MAXFONTS * sizeof(XmString *));
      for(i = 0; i < MAXFONTS; i++)
        font_list[i] =
             XmStringCreateSimple(graphic_list.font_name(i + 1));
      list_box =
              XmCreateScrolledList(drawing_a, "Fonts", NULL, 0);
      XtVaSetValues(list_box,
        XmNitems, font_list,
        XmNitemCount, MAXFONTS,
        XmNvisibleItemCount, 7,
        NULL);
```

```
      for(i = 0; i < MAXFONTS; i++)
        XmStringFree(font_list[i]);
      XtFree((char *) font_list);
      XtAddCallback(list_box, XmNdefaultActionCallback,
                    font_list_cb, NULL);
      XtManageChild(list_box);
      break;
    case 2:
      if(selected_object_ptr != NULL) {
        selected_object_ptr->unselect();
        selected_object_ptr = NULL;
      }
      graphic_list.get_del_op_list(del_op_str, del_op_id,
                                   num_del_ops);
      op_list = (XmStringTable)
                XtMalloc((num_del_ops + 1) * sizeof(XmString *));
      for(i = 0; i < num_del_ops; i++)
        op_list[i] = XmStringCreateSimple(del_op_str[i]);
      op_list[num_del_ops] = XmStringCreateSimple("Cancel");
      op_box = XmCreateScrolledList(drawing_a, "Undelete",
                                    NULL, 0);
      XtVaSetValues(op_box,
                    XmNitems, op_list,
                    XmNitemCount, num_del_ops + 1,
                    XmNvisibleItemCount, 7,
                    NULL);
      for(i = 0; i < num_del_ops + 1; i++)
        XmStringFree(op_list[i]);
      XtFree((char *) op_list);
      XtAddCallback(op_box, XmNdefaultActionCallback,
                    op_list_cb, NULL);
      XtManageChild(op_box);
      break;
    default:
      break;
  }
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
}

//   This function is called by the window manager when a
// menu option from the 'Draw Options' pulldown menu is
// selected.  Its address is passed as a parameter to X when
// the menus are defined.

static void draw_options_cb(Widget, XtPointer client_data,
                            XtPointer) {
  int item_no = (int) client_data;

  handle_draw_options(item_no);
}

//   This function is called by the window manager when a
// menu option from the 'Help' pulldown menu is
// selected.  Its address is passed as a parameter to X when
```

```
// the menus are defined.


static void help_cb(Widget, XtPointer, XtPointer) {};

/*  ================= Not USED !!!! ====================

void set_color(Widget widget, char *color) {
  Display *dpy = XtDisplay(widget);
  Colormap cmap = DefaultColormapOfScreen(XtScreen(widget));
  XColor col, unused;

  if(!XAllocNamedColor(dpy, cmap, color, &col, &unused)) {
    error_box("Can't allocate color");
    return;
  }
  XSetForeground(dpy, std_graphics_context, col.pixel);
}
*/

//   Builds the top line menu bar.

void build_menu_bar(Widget &main_w, Widget &menubar) {
  XmString graph, draw_options, color_options, help, save, print,
    decompose, edit_parent, return_sde, font_options,
    undelete_operator, restore, quit_no_save;
  Widget widget;

  graph = XmStringCreateSimple("Graph");
  color_options = XmStringCreateSimple("Color");
  draw_options = XmStringCreateSimple("Draw Options");
  help = XmStringCreateSimple("Help");
  save = XmStringCreateSimple("Save and Continue");
  restore = XmStringCreateSimple("Restore From Save");
  print = XmStringCreateSimple("Print");
  decompose = XmStringCreateSimple("Decompose");
  edit_parent = XmStringCreateSimple("Edit Parent");
  return_sde = XmStringCreateSimple("Return to SDE");
  quit_no_save = XmStringCreateSimple("Quit Without Saving");
  font_options = XmStringCreateSimple("Font");
  undelete_operator = XmStringCreateSimple("Undelete Operator");

  menubar = XmVaCreateSimpleMenuBar(main_w, "menubar",
      XmVaCASCADEBUTTON, graph, NULL,
      XmVaCASCADEBUTTON, draw_options, NULL,
      XmVaCASCADEBUTTON, help, 'H', NULL);
  if(widget = XtNameToWidget(menubar, "button_2"))
    XtVaSetValues(menubar, XmNmenuHelpWidget, widget, NULL);
  XmVaCreateSimplePulldownMenu(menubar, "graph_menu", 0,
                               graph_cb,
    XmVaPUSHBUTTON, save, 'S', NULL, NULL,
    XmVaPUSHBUTTON, restore, 'e', NULL, NULL,
    XmVaPUSHBUTTON, print, 'P', NULL, NULL,
    XmVaPUSHBUTTON, decompose, 'D', NULL, NULL,
```

```
    XmVaPUSHBUTTON, edit_parent, 'i', NULL, NULL,
    XmVaPUSHBUTTON, return_sde, 'R', NULL, NULL,
    XmVaPUSHBUTTON, quit_no_save, 'Q', NULL, NULL,
    NULL);
  XmVaCreateSimplePulldownMenu(menubar, "draw_options", 1,
                               draw_options_cb,
    XmVaPUSHBUTTON, color_options, 'C', NULL, NULL,
    XmVaPUSHBUTTON, font_options; 'F', NULL, NULL,
    XmVaPUSHBUTTON, undelete_operator, 'U', NULL, NULL,
    NULL);
  XmVaCreateSimplePulldownMenu(menubar, "help_menu", 2, help_cb,
    XmVaPUSHBUTTON, help, 'H', NULL, NULL,
    NULL);
  XmStringFree(graph);
  XmStringFree(draw_options);
  XmStringFree(color_options);
  XmStringFree(help);
  XmStringFree(save);
  XmStringFree(restore);
  XmStringFree(print);
  XmStringFree(decompose);
  XmStringFree(edit_parent);
  XmStringFree(quit_no_save);
  XmStringFree(return_sde);
  XmStringFree(font_options);
  XmStringFree(undelete_operator);
}

//   Creates the push buttons used to select the tools.

void make_buttons(Widget &rowcol, Widget &op_button,
                  Widget &term_button, Widget &stream_button,
                  Widget &property_button,
                  Widget &select_button,
                  Pixmap &op_button_pixmap,
                  Pixmap &term_button_pixmap,
                  Pixmap &stream_button_pixmap,
                  Pixmap &property_button_pixmap,
                  Pixmap &select_button_pixmap,
                  Display *display_ptr, Screen *screen_ptr) {
  Window root_window = RootWindowOfScreen(screen_ptr);
  unsigned int screen_depth = DefaultDepthOfScreen(screen_ptr);

  op_button_pixmap = XCreatePixmap(display_ptr, root_window,
                     BUTTONWIDTH, BUTTONWIDTH, screen_depth);
  term_button_pixmap = XCreatePixmap(display_ptr, root_window,
                       BUTTONWIDTH, BUTTONWIDTH, screen_depth);
  stream_button_pixmap = XCreatePixmap(display_ptr, root_window,
                       BUTTONWIDTH, BUTTONWIDTH, screen_depth);
  property_button_pixmap =
    XCreatePixmap(display_ptr, root_window,
                       BUTTONWIDTH, BUTTONWIDTH, screen_depth);
  select_button_pixmap = XCreatePixmap(display_ptr, root_window,
                       BUTTONWIDTH, BUTTONWIDTH, screen_depth);
```

```
XFillRectangle(display_ptr, (Drawable) op_button_pixmap,
               erase_context, 0, 0, BUTTONWIDTH, BUTTONWIDTH);
XFillRectangle(display_ptr, (Drawable) term_button_pixmap,
               erase_context, 0, 0, BUTTONWIDTH, BUTTONWIDTH);
XFillRectangle(display_ptr, (Drawable) stream_button_pixmap,
               erase_context, 0, 0, BUTTONWIDTH, BUTTONWIDTH);
XFillRectangle(display_ptr, (Drawable) property_button_pixmap,
               erase_context, 0, 0, BUTTONWIDTH, BUTTONWIDTH);
XFillRectangle(display_ptr, (Drawable) select_button_pixmap,
               erase_context, 0, 0, BUTTONWIDTH, BUTTONWIDTH);
XSetLineAttributes(display_ptr, std_graphics_context, 2,
                   LineSolid, CapButt, JoinMiter);
XDrawArc(display_ptr, (Drawable) op_button_pixmap,
         std_graphics_context, 10, 10, 45, 45, CIRCLE_BEGIN,
         FULL_CIRCLE);
XDrawRectangle(display_ptr, (Drawable) term_button_pixmap,
               std_graphics_context, 10, 10, 45, 45);
XDrawLine(display_ptr, (Drawable) stream_button_pixmap,
          std_graphics_context, 10, 10, 55, 55);
XDrawString(display_ptr, (Drawable) property_button_pixmap,
            std_graphics_context, 5, 35, "Properties", 10);
XDrawString(display_ptr, (Drawable) select_button_pixmap,
            std_graphics_context, 10, 35, "Select", 6);
op_button = XtVaCreateManagedWidget("op_button",
                            xmDrawnButtonWidgetClass,
                            rowcol,
                            XmNrecomputeSize, FALSE,
                            XmNpushButtonEnabled, TRUE,
                            XmNwidth, BUTTONWIDTH,
                            XmNheight, BUTTONWIDTH,
                            XmNlabelType, XmPIXMAP,
                            XmNlabelPixmap, op_button_pixmap,
                            NULL);
term_button = XtVaCreateManagedWidget("term_button",
                           xmDrawnButtonWidgetClass,
                           rowcol,
                           XmNrecomputeSize, FALSE,
                           XmNpushButtonEnabled, TRUE,
                           XmNwidth, BUTTONWIDTH,
                           XmNheight, BUTTONWIDTH,
                           XmNlabelType, XmPIXMAP,
                           XmNlabelPixmap, term_button_pixmap,
                           NULL);
stream_button = XtVaCreateManagedWidget("stream_button",
                          xmDrawnButtonWidgetClass,
                          rowcol,
                          XmNrecomputeSize, FALSE,
                          XmNpushButtonEnabled, TRUE,
                          XmNwidth, BUTTONWIDTH,
                          XmNheight, BUTTONWIDTH,
                          XmNlabelType, XmPIXMAP,
                          XmNlabelPixmap, stream_button_pixmap,
                          NULL);
```

```
  property_button = XtVaCreateManagedWidget("property_button",
                          xmDrawnButtonWidgetClass,
                          rowcol,
                          XmNrecomputeSize, FALSE,
                          XmNpushButtonEnabled, TRUE,
                          XmNwidth, BUTTONWIDTH,
                          XmNheight, BUTTONWIDTH,
                          XmNlabelType, XmPIXMAP,
                          XmNlabelPixmap, property_button_pixmap,
                          NULL);

  select_button = XtVaCreateManagedWidget("select_button",
                            xmDrawnButtonWidgetClass,
                            rowcol,
                            XmNrecomputeSize, FALSE,
                            XmNpushButtonEnabled, TRUE,
                            XmNwidth, BUTTONWIDTH,
                            XmNheight, BUTTONWIDTH,
                            XmNlabelType, XmPIXMAP,
                            XmNlabelPixmap, select_button_pixmap,
                            NULL);
  XSetLineAttributes(display_ptr, std_graphics_context, 1,
                     LineSolid, CapButt, JoinMiter);

}

//   Redraws the drawing canvas.

void redraw(Widget, XtPointer,
            XtPointer cbs) {
  XmDrawingAreaCallbackStruct *temp_ptr;

  temp_ptr = (XmDrawingAreaCallbackStruct *) cbs;
  XCopyArea(temp_ptr->event->xexpose.display,
            drawing_area_pixmap, temp_ptr->window,
            std_graphics_context, 0, 0, width, height, 0, 0);
}

//   Draws a square black box on the canvas to aid in
// graphic manipulation of objects.

void draw_handle(GC graphics_context, int x, int y) {

  x -= HANDLESIZE / 2;
  y -= HANDLESIZE / 2;
  if(x < 0)
    x = 0;
  if(y < 0)
    y = 0;

//  When the display function is set to GXxor, the pixel being
// written is exclusive-or'ed with the target pixel to
// determine color.  This means that writing the same pixel with
// the same color twice restores the original color, simplifying
```

```
// the process of erasing handles.

  XSetFunction(display_ptr, graphics_context, GXxor);
  XFillRectangle(display_ptr, draw_window, graphics_context,
                 x, y, HANDLESIZE, HANDLESIZE);
  XFillRectangle(display_ptr, drawing_area_pixmap,
                 graphics_context,
                 x, y, HANDLESIZE, HANDLESIZE);
  XSetFunction(display_ptr, graphics_context, GXcopy);
}

//   This function erases the temporary guidelines used when
// streams are drawn.
//   Dotted lines are erased first, then handles.  Since each
// handle is overwritten with the following dotted line, an
// erased handle makes an erased blotch in the beginning of the
// next segment.   When the next segment is written in xor mode,
// it makes a black mark where the erased handle overwrote the
// beginning of its segment.

void erase_guides(int from_stream_id, SplineObject temp_spline) {
  OperatorObject *temp_operator_ptr;
  XYPAIR line_start, line_end;

  temp_spline.reset_iter();
  if(from_stream_id != 0) {
    temp_operator_ptr = (OperatorObject *)
      graphic_list.target_object(OPERATOROBJECT, from_stream_id);
    line_start = temp_operator_ptr->center();
  }
  else
    line_start = temp_spline.next_pair();
  line_end = temp_spline.next_pair();
  while(line_end.x != -1) {
    XDrawLine(display_ptr, draw_window, dotted_context,
              line_start.x, line_start.y, line_end.x,
              line_end.y);
    XDrawLine(display_ptr,drawing_area_pixmap, dotted_context,
              line_start.x, line_start.y, line_end.x,
              line_end.y);
    line_start = line_end;
    line_end = temp_spline.next_pair();
  }
  temp_spline.reset_iter();
  line_end = temp_spline.next_pair();
  while(line_end.x != -1) {
    draw_handle(std_graphics_context, line_end.x, line_end.y);
    line_end = temp_spline.next_pair();
  }
}

//   This function is called when a stream is being drawn
// and the mouse is clicked on either a clear spot on the
// drawing canvas, or on top of another stream.  If a double-
```

```
// click is registered, the user wants to terminate an external
// stream.

void handle_null_point(int from_stream_id, int &last_point_x,
                       int &last_point_y,
                       int &x_state, int &y_state,
                       XEvent in_event,
                       SplineObject &temp_spline, BOOLEAN &done,
                       GraphObject *&temp_object_ptr,
                       StreamObject *&temp_stream_ptr) {

//   Checks for two clicks in the same spot.

  if((from_stream_id != 0) &&
     ((last_point_x - (HANDLESIZE / 2) - HITFUDGE)
      < in_event.xbutton.x) &&
     ((last_point_x + (HANDLESIZE / 2) + HITFUDGE)
      > in_event.xbutton.x) &&
     ((last_point_y - (HANDLESIZE / 2) - HITFUDGE)
      < in_event.xbutton.y) &&
     ((last_point_y + (HANDLESIZE / 2) + HITFUDGE)
      > in_event.xbutton.y)) {
    erase_guides(from_stream_id, temp_spline);
    int new_id = graphic_list.request_id(STREAMOBJECT);
    temp_stream_ptr = new StreamObject("", new_id,
                                       from_stream_id,
                                       0, 0, temp_spline, TRUE,
                                       FALSE);
    temp_stream_ptr->set_object_ptrs(&graphic_list);
    graphic_list.add(temp_stream_ptr);
    temp_stream_ptr->draw(SOLID);
    temp_stream_ptr = NULL;
    temp_object_ptr = NULL;
    done = TRUE;
    temp_spline.clear();
  }
  else {
    x_state = in_event.xbutton.x;
    y_state = in_event.xbutton.y;
    temp_spline.add(x_state, y_state);
    XDrawLine(display_ptr, draw_window, dotted_context,
              last_point_x, last_point_y, x_state, y_state);
    XDrawLine(display_ptr, drawing_area_pixmap, dotted_context,
              last_point_x, last_point_y, x_state, y_state);
    draw_handle(std_graphics_context, x_state, y_state);
#ifdef GE_DEBUG
    cout << "ge: " << x_state << " " << y_state << " " <<
      HANDLESIZE << endl;
#endif
    last_point_x = x_state;
    last_point_y = y_state;
  }
}
```

```
//   Once the user selects the Stream Tool and begins to draw,
// the draw_stream() function handles all events to speed up
// performance.

void draw_stream(int initial_x, int initial_y) {
  GraphObject *temp_object_ptr;
  OperatorObject *conv_ptr;
  XYPAIR temp_pair;
  int from_stream_id, x_state, y_state, last_point_x,
    last_point_y;
  unsigned long stream_event_mask =
                          (ButtonPressMask | PointerMotionMask);
  unsigned long normal_mask = (ButtonPressMask | KeyPressMask |
                               ButtonMotionMask | ExposureMask |
                               ButtonReleaseMask);
  XEvent in_event;
  StreamObject *temp_stream_ptr;
  SplineObject temp_spline;
  BOOLEAN done = FALSE;

  temp_object_ptr = graphic_list.hit(initial_x, initial_y);
  if(temp_object_ptr == NULL) {  // External stream
    from_stream_id = 0;
    temp_spline.add(initial_x, initial_y);
    x_state = initial_x;
    y_state = initial_y;
    draw_handle(std_graphics_context, x_state, y_state);
  }
  else {
    if(temp_object_ptr->is_a() != OPERATOROBJECT) {
  //  External Stream
      from_stream_id = 0;
      temp_spline.add(initial_x, initial_y);
      x_state = initial_x;
      y_state = initial_y;
      draw_handle(std_graphics_context, x_state, y_state);
    }
    else {
      conv_ptr = (OperatorObject *) temp_object_ptr;
      from_stream_id = conv_ptr->id();
      temp_object_ptr = NULL;
      temp_pair = conv_ptr->center();
      x_state = temp_pair.x;
      y_state = temp_pair.y;
    }
  }
  last_point_x = x_state;
  last_point_y = y_state;
  XSelectInput(display_ptr, draw_window,
               stream_event_mask);
  while(done == FALSE) {  // monitors the event loop
    XNextEvent(display_ptr, &in_event);
    if(in_event.xbutton.window == draw_window) {
      switch(in_event.type) {
```

```
        case MotionNotify:

#ifdef GE_DEBUG
          cout << "Motion" << endl;
#endif

          XDrawLine(display_ptr, draw_window, dotted_context,
                  last_point_x, last_point_y, x_state, y_state);
          XDrawLine(display_ptr, drawing_area_pixmap,
                    dotted_context, last_point_x, last_point_y,
                    x_state, y_state);
          x_state = in_event.xbutton.x;
          y_state = in_event.xbutton.y;
          XDrawLine(display_ptr, draw_window, dotted_context,
                  last_point_x, last_point_y, x_state, y_state);
          XDrawLine(display_ptr, drawing_area_pixmap,
                    dotted_context, last_point_x, last_point_y,
                    x_state, y_state);
          break;
        case ButtonPress:
#ifdef GE_DEBUG
          cout << "buttonpress" << endl;
#endif
          XDrawLine(display_ptr, draw_window, dotted_context,
                  last_point_x, last_point_y, x_state, y_state);
          XDrawLine(display_ptr, drawing_area_pixmap,
                    dotted_context, last_point_x, last_point_y,
                    x_state, y_state);
          temp_object_ptr = graphic_list.hit(in_event.xbutton.x,
                                             in_event.xbutton.y);
          if(temp_object_ptr == NULL) {
            handle_null_point(from_stream_id, last_point_x,
                              last_point_y, x_state, y_state,
                              in_event, temp_spline, done,
                              temp_object_ptr, temp_stream_ptr);
          }
          else
            if(temp_object_ptr->is_a() == OPERATOROBJECT) {
              erase_guides(from_stream_id, temp_spline);
              int new_id = graphic_list.request_id(STREAMOBJECT);
              temp_stream_ptr =
                   new StreamObject("", new_id, from_stream_id,
                                    temp_object_ptr->id(), 0,
                                    temp_spline, TRUE, FALSE);
              temp_stream_ptr->set_object_ptrs(&graphic_list);
              graphic_list.add(temp_stream_ptr);
              temp_stream_ptr->draw(SOLID);
              temp_stream_ptr = NULL;
              temp_object_ptr = NULL;
              done = TRUE;
              temp_spline.clear();
            }
            else
              if(temp_object_ptr->is_a() == STREAMOBJECT) {
```

```
                handle_null_point(from_stream_id, last_point_x,
                                  last_point_y, x_state, y_state,
                                  in_event, temp_spline, done,
                                  temp_object_ptr, temp_stream_ptr);
              }
            break;
        }  //switch
      } //if right window
    }  //while done == FALSE
    done = FALSE;
    XSelectInput(display_ptr, draw_window, normal_mask);
}

//   Draws the outline of the text being moved.

void draw_text_shadow(int x, int y, int width, int height) {

    XDrawRectangle(display_ptr, drawing_area_pixmap,
                   dotted_context, x - width / 2, y - height / 2,
                   width, height);
    XDrawRectangle(display_ptr, draw_window, dotted_context,
                   x - width / 2, y - height / 2, width, height);
}

//   The main draw routine.  This function is called by the
// window manager every time the mouse is moved, a mouse button
// pressed, or a key pressed inside the draw window.  It is
// called with a string token that indicates why it was called,
// and processes the event accordingly.

void draw(Widget, XEvent *event, String *args, Cardinal *) {
    static OperatorObject *temp_operator_ptr = NULL;
    static StreamObject *temp_stream_ptr = NULL;
    static BOOLEAN first_draw = TRUE, handle_selected = FALSE,
      text_selected = FALSE, drawing_changed = FALSE;
    static int x_state, y_state, shadow_height, shadow_width,
      from_stream_id;
    GraphObject *temp_object_ptr = NULL;
    int x = event->xbutton.x;
    int y = event->xbutton.y;
    OperatorObject *conv_op_ptr;
    SplineObject temp_spline;
    StreamObject *conv_st_ptr;

#ifdef GE_DEBUG
    cout << args[0] << endl;
#endif

    if(strcmp(args[0], "down") == 0) {  // Button pressed
      XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
      x_state = x;
      y_state = y;
      if(tool_state == SELECT_TOOL) {
        if(selected_object_ptr != NULL) {
```

```
    if(selected_object_ptr->hit_handle(x, y)) {
      handle_selected = TRUE;
      if(selected_object_ptr->is_a() == OPERATOROBJECT) {
        delete temp_operator_ptr;
        conv_op_ptr = (OperatorObject *) selected_object_ptr;
        temp_operator_ptr =
          new OperatorObject("", 0, 0, conv_op_ptr->x(),
                             conv_op_ptr->y(),
                             conv_op_ptr->radius(),
                             default_color, FALSE,
                             conv_op_ptr->is_composite(),
                             conv_op_ptr->is_terminator());
        temp_operator_ptr->set_handle_selected(
                             conv_op_ptr->handle_selected());
      }
    }
    else {          // Unselects previously selected object
      handle_selected = FALSE;
      selected_object_ptr->unselect();
      selected_object_ptr = NULL;
      delete temp_operator_ptr;
      temp_operator_ptr = NULL;
    }
  }
  if(handle_selected == FALSE) {
    temp_object_ptr = graphic_list.hit(x, y);
    if(temp_object_ptr != NULL) {
      temp_object_ptr->select();
      selected_object_ptr = temp_object_ptr;
      text_selected = selected_object_ptr->text_selected();
      if(temp_object_ptr->is_a() == OPERATOROBJECT) {
          // Makes temporary operator to move around
        delete temp_operator_ptr;
        conv_op_ptr = (OperatorObject *) temp_object_ptr;
        temp_operator_ptr =
          new OperatorObject("", 0, 0, conv_op_ptr->x(),
                             conv_op_ptr->y(),
                             conv_op_ptr->radius(),
                             default_color, FALSE,
                             conv_op_ptr->is_composite(),
                             conv_op_ptr->is_terminator());
      }
    }
    else { // No object selected
      temp_object_ptr = NULL;
      selected_object_ptr = NULL;
      delete temp_operator_ptr;
      temp_operator_ptr = NULL;
    }
  }
}
else {  // button down, operator tool selected?
  if((tool_state == OPERATOR_TOOL) ||
     (tool_state == TERMINATOR_TOOL)) {
```

```
      int new_id = graphic_list.request_id(OPERATOROBJECT);
      if(tool_state == OPERATOR_TOOL) {
        temp_operator_ptr =
          new OperatorObject("", new_id, 0, 0, 0, 20,
                             default_color, TRUE, FALSE,
                             FALSE);
        temp_operator_ptr->set_location(x, y);
      }
      else
        if(tool_state == TERMINATOR_TOOL) {
          temp_operator_ptr =
            new OperatorObject("", new_id, 0, 0, 0, 20,
                               default_color, TRUE, FALSE,
                               TRUE);
          temp_operator_ptr->set_location(x, y);
        }
      graphic_list.add((GraphObject *) temp_operator_ptr);
      temp_operator_ptr->draw(SOLID);
      temp_operator_ptr = NULL;
    }
    else // button down, stream tool selected?
      if(tool_state == STREAM_TOOL) {
        draw_stream(x, y);
      }
  }
}
else // button not down
  if(strcmp(args[0], "motion") == 0) {
    if(tool_state == SELECT_TOOL)
      if(selected_object_ptr != NULL) {
        drawing_changed = TRUE;
        if(text_selected) {
          if(first_draw == TRUE) {
            shadow_width = selected_object_ptr->text_width();
            shadow_height = selected_object_ptr->text_height();
            draw_text_shadow(x, y, shadow_width,
                             shadow_height);
            first_draw = FALSE;
          }
          else {
            draw_text_shadow(x_state, y_state, shadow_width,
                             shadow_height);
            draw_text_shadow(x, y, shadow_width,
                             shadow_height);
          }
        }
        else
          if(selected_object_ptr->is_a() == OPERATOROBJECT) {
            if(handle_selected == TRUE) {

              if(first_draw == TRUE) {
                selected_object_ptr->erase();
                selected_object_ptr->unselect();
                selected_object_ptr->draw(SOLID);
```

```
                temp_operator_ptr->draw(DOTTED);
                first_draw = FALSE;
              }
              else {
                temp_operator_ptr->move_handle(x - x_state,
                                               y - y_state);
//                cout << "x: " << x << " y" << y << " xstate: " << x_state <<
" y_state: " << y_state << endl;
              }
            }
            else {
              if(first_draw == TRUE)

//   Drawing the same thing twice in xor mode erases it. When
// moving an object, it is drawn once the first and last time,
// and twice afterwards

                first_draw = FALSE;
              else
                temp_operator_ptr->draw(DOTTED);
              temp_operator_ptr->move(x - x_state, y - y_state);
              temp_operator_ptr->draw(DOTTED);
            }
          }
          else {
            if(selected_object_ptr->is_a() == STREAMOBJECT) {
              if(handle_selected) {
                if(first_draw == TRUE) {
                  conv_st_ptr =
                         (StreamObject *) selected_object_ptr;
                  conv_st_ptr->erase_handle();
                  draw_handle(std_graphics_context, x, y);
                  first_draw = FALSE;
                }
                else {
                  draw_handle(std_graphics_context, x_state,
                              y_state);
                  draw_handle(std_graphics_context, x, y);
                  selected_object_ptr->move_handle(
                                               x - x_state,
                                               y - y_state);
                }
              }
            }
          }
        x_state = x;
        y_state = y;
      }
    }
    else // button not down, mouse not moved
      if(strcmp(args[0], "up") == 0) {
        if(tool_state == SELECT_TOOL)
          if(selected_object_ptr != NULL) {
            if(text_selected) {
```

```
            if(first_draw == FALSE) {
              draw_text_shadow(x_state, y_state,
                               shadow_width, shadow_height);
              selected_object_ptr->text_locate(x, y);
            }
          }
          else
            if(selected_object_ptr->is_a() == OPERATOROBJECT) {
              if(first_draw == FALSE) {
                temp_operator_ptr->draw(DOTTED);
                XYPAIR temp_pair = temp_operator_ptr->center();
                conv_op_ptr =
                  (OperatorObject *) selected_object_ptr;
                conv_op_ptr->set_radius(
                                 temp_operator_ptr->radius());
                conv_op_ptr->set_location(temp_pair.x,
                                          temp_pair.y);
                if(handle_selected)
                  conv_op_ptr->set_default_text_location();
              }
            }
            else
              if((selected_object_ptr->is_a() == STREAMOBJECT)
                 && (handle_selected)) {
                draw_handle(std_graphics_context, x_state,
                            y_state);
              }
            if(drawing_changed == TRUE) {
              graphic_list.move_notify(
                           selected_object_ptr->is_a(),
                           selected_object_ptr->id());
              graphic_list.draw();
              drawing_changed = FALSE;
            }
            handle_selected = FALSE;
          }
        first_draw = TRUE;
      }
      else {

#ifdef GE_DEBUG
        cout << args[0] << endl;
        if(selected_object_ptr == NULL)
          cout << "selected_object_ptr == NULL" << endl;
#endif

        if(strcmp(args[0], "key") == 0) {

#ifdef GE_DEBUG
         cout << "key pressed: " << event->xkey.keycode << endl;
#endif

          if(alt_selected) {          // alt key pressed
            alt_selected = FALSE;
```

```
  switch(event->xkey.keycode) {
  case 85:
    handle_graph_options(0);
    break;
  case 63:
    handle_graph_options(1);
    break;
  case 70:
    handle_graph_options(2);
    break;
  case 86:
    handle_graph_options(3);
    break;
  case 68:
    handle_graph_options(4);
    break;
  case 64:
    handle_graph_options(5);
    break;
  case 61:
    handle_graph_options(6);
    break;
  case 109:
    handle_draw_options(0);
    break;
  case 87:
    handle_draw_options(1);
    break;
  case 67:
    handle_draw_options(2);
    break;
  case 26:
    alt_selected = TRUE;
    break;
  default:
    break;
  }
}
else
 if(event->xkey.keycode == 26)
   alt_selected = TRUE;
 else
   if(selected_object_ptr != NULL) {
     if((event->xkey.keycode == 50) || // backspace key
        (event->xkey.keycode == 73)) { // delete key
       selected_object_ptr->erase();
       int deleted_op_id = selected_object_ptr->id();
       graphic_list.delete_notify(selected_object_ptr->
                                  is_a(), deleted_op_id);
       selected_object_ptr->set_deleted();
       selected_object_ptr = NULL;
       graphic_list.draw();
     }
   }
```

```
        }
      }
}

//   Checks to see whether the indicated string
// represents a valid number.

BOOLEAN valid_num_string(char *num) {
  int index, num_length;

  if(num != NULL) {
    num_length = strlen(num);
    if(num_length > 0) {
      for(index = 0; index < num_length; index++) {
        if((num[index] != '\n') &&
           (num[index] != '-') &&
           ((num[index] < '0') || (num[index] > '9')))
          return FALSE;
      }
      return TRUE;
    }
  }
  return FALSE;
}

//   Callback function.  Just destroys the widget.

void widget_killer(Widget widget, XtPointer, XtPointer) {

  XtDestroyWidget(widget);
}

//   Callback function called after user has entered data
// in the property dialog box.  Function reads the data from
// the dialog box and processes it accordingly.

void read_property(Widget widget, XtPointer clientdata,
                   XtPointer cbs) {
  XmSelectionBoxCallbackStruct *temp_ptr;
  char *text;
  Widget* widget_ptr;
  StreamObject *conv_ptr;

  selected_object_ptr->erase(); // so it can be redrawn
  widget_ptr = (Widget *) clientdata;
  text = XmTextFieldGetString(*widget_ptr);
  if(text != NULL)
    selected_object_ptr->replace_name(text);
  temp_ptr = (XmSelectionBoxCallbackStruct *) cbs;
  XmStringGetLtoR(temp_ptr->value, XmSTRING_DEFAULT_CHARSET,
                  &text);
  if(valid_num_string(text))
    selected_object_ptr->set_constraint(atoi(text));
  else
```

```
    selected_object_ptr->set_constraint(NULL_VALUE);
  if(selected_object_ptr->is_a() == STREAMOBJECT) {
    conv_ptr = (StreamObject *) selected_object_ptr;
    conv_ptr->set_state_variable(state_stream);
  }
  selected_object_ptr->draw(SOLID);
  XtDestroyWidget(widget);
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
}

//   Callback function.  Called when Operator Tool button is
// pressed.

void op_button_cb(Widget, XtPointer, XtPointer) {

  tool_state = OPERATOR_TOOL;
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  if(selected_object_ptr != NULL) {
    selected_object_ptr->unselect();
    selected_object_ptr = NULL;
  }
  XtVaSetValues(tool_indicator, XmNvalue, "Operator Tool", NULL);
}

//   Callback function.  Called when Terminator Tool button is
// pressed.

void term_button_cb(Widget, XtPointer, XtPointer) {

  tool_state = TERMINATOR_TOOL;
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  if(selected_object_ptr != NULL) {
    selected_object_ptr->unselect();
    selected_object_ptr = NULL;
  }
  XtVaSetValues(tool_indicator, XmNvalue, "Terminator Tool",
                NULL);
}

//   Callback function.  Called when Stream Tool button is
// pressed.

void stream_button_cb(Widget, XtPointer, XtPointer) {

  tool_state = STREAM_TOOL;
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  if(selected_object_ptr != NULL) {
    selected_object_ptr->unselect();
    selected_object_ptr = NULL;
  }
  XtVaSetValues(tool_indicator, XmNvalue, "Stream Tool", NULL);
}

//   Callback function.  Called when Select Tool button is
```

```
// pressed.

void select_button_cb(Widget, XtPointer, XtPointer) {

  tool_state = SELECT_TOOL;
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  XtVaSetValues(tool_indicator, XmNvalue, "Select Tool", NULL);
}

//   Null Callback.

void null_cb(Widget, XtPointer, XtPointer) {}

//   Callback function.  Called when the radio buttons in the
// properties dialog box are pushed.  Called twice: once to
// unselect old button, again to select the new one.

void radio_box_cb(Widget, XtPointer which,
                  XtPointer cbs) {
  XmToggleButtonCallbackStruct *state =
    (XmToggleButtonCallbackStruct *) cbs;

  if(state->set) {
    if((int) which == 0)
      state_stream = FALSE;
    else
      state_stream = TRUE;
  }
}

//   Called when Properties button is selected.

void property_button_cb(Widget widget, XtPointer, XtPointer) {

  static Widget dialog, operator_name;
  XmString t, button1, button2;
  Arg args[3];
  Widget rc, radio_box;
  char *prompt, buffer[INPUT_LINE_SIZE];
  int initial_button, constraint;
  StreamObject *conv_ptr;
  XmString init_value;

  if(selected_object_ptr != NULL) {
    if(selected_object_ptr->is_a() == OPERATOROBJECT)
      t = XmStringCreateSimple("Enter Maximum Execution Time:");
    else
      if(selected_object_ptr->is_a() == STREAMOBJECT)
        t = XmStringCreateSimple("Enter Latency:");
    selected_object_ptr->set_modified();
    XtSetArg(args[0], XmNselectionLabelString, t);
    XtSetArg(args[1], XmNautoUnmanage, False);
    XtSetArg(args[2], XmNuserData, 0);
    dialog = XmCreatePromptDialog(widget, "Properties", args, 3);
```

```
  XmStringFree(t);
  if(selected_object_ptr->is_a() == OPERATOROBJECT)
    prompt = strdup("Operator Name:");
  else
    if(selected_object_ptr->is_a() == STREAMOBJECT)
      prompt = strdup("Stream Name:");

  rc = XtVaCreateWidget("rowcol", xmRowColumnWidgetClass,
                        dialog, NULL);
  XtVaCreateManagedWidget(prompt, xmLabelGadgetClass, rc,
                        XmNalignment, XmALIGNMENT_BEGINNING,
                        NULL);
  operator_name =
    XtVaCreateManagedWidget("text_field",
                          xmTextFieldWidgetClass, rc, NULL);
    //null callback added to make user use ok button or second box.
  XtVaSetValues(operator_name, XmNvalue,
                                    selected_object_ptr->name(), NULL);
  if(selected_object_ptr->is_a() == STREAMOBJECT) {
    conv_ptr = (StreamObject *) selected_object_ptr;
    if(conv_ptr->is_state_variable())
      initial_button = 1;
    else
      initial_button = 0;
    button1 = XmStringCreateSimple("Not State Stream");
    button2 = XmStringCreateSimple("State Stream");
    radio_box = XmVaCreateSimpleRadioBox(rc, "radio_box",
                  initial_button, radio_box_cb,
                  XmVaRADIOBUTTON, button1, NULL, NULL, NULL,
                  XmVaRADIOBUTTON, button2, NULL, NULL, NULL,
                  NULL);
    XmStringFree(button1);
    XmStringFree(button2);
    XtManageChild(radio_box);
  }
  constraint = selected_object_ptr->constraint();
  if(constraint != 0) {
    sprintf(buffer, "%d", constraint);
    init_value = XmStringCreateSimple(buffer);
    XtVaSetValues(dialog, XmNtextString, init_value, NULL);
    XmStringFree(init_value);
  }
  XtAddCallback(operator_name, XmNactivateCallback, null_cb,
                NULL);
  XtAddCallback(dialog, XmNokCallback, read_property,
                (XtPointer) &operator_name);
  XtAddCallback(dialog, XmNcancelCallback, widget_killer,
                NULL);
  XtUnmanageChild(XmSelectionBoxGetChild(dialog,
                  XmDIALOG_HELP_BUTTON)); // Hide help button.
  XtManageChild(rc);
  XtManageChild(dialog);
  XtPopup(XtParent(dialog), XtGrabNone);
}
```

```
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
}

//   If graph_editor is invoked in viewer mode, this function
// handles ClientMessage events from the syntax-directed editor.
// Commented-out code handles data passed in a property, which
// this version of the editor doesn't take advantage of.
// Used during testing, and left-in for future use, if necessary.

void event_handler(Widget widget, XtPointer,
                   XEvent* in_event, Boolean*) {
  Display *display_ptr = XtDisplayOfObject(widget);
  Window window = DefaultRootWindow(display_ptr);
//  char **data;
//  int return_count;
//  XTextProperty text_prop_return;
//  Atom property_name;
  char message_in[30];

#ifdef GE_DEBUG
  cout << "\nReceiving:\n" << endl;
#endif

  strcpy(message_in, in_event->xclient.data.b);
  if(strcmp(message_in, "GEDATAIN") == 0) {
//    property_name = XInternAtom(display_ptr, "CAPS_DATA", False);
//    XGetTextProperty(display_ptr, window, &text_prop_return, property_name);
//    XTextPropertyToStringList(&text_prop_return, &data, &return_count);
//    if(strcmp(data[0], "Out!!!") == 0)
//      exit(0);
//    else {
//      display_string = XmStringCreateSimple(data[0]);
//      XtVaSetValues(*((Widget *) label), XmNlabelString, display_string,
NULL);
//      XmStringFree(display_string);
//    }

  graphic_list.build_from_disk();
  graphic_list.draw();
  }
  else
    if(strcmp(message_in, "GEQUIT") == 0) {

#ifdef GE_DEBUG
      cout << "Got viewer quit" << endl;
#endif

      exit(0);
    }
}

//   Displays information about event received by program.
// Not currently used, but left in for future use.
```

```
void event_list(XEvent *in_event) {

  cout << "type: " << in_event->xclient.type;
  cout << "\nserial: " << in_event->xclient.serial;
  cout << "\nsend_event: " << in_event->xclient.send_event;
  cout << "\ndisplay: " << in_event->xclient.display;
  cout << "\nwindow: " << in_event->xclient.window;
  cout << "\nformat: " << in_event->xclient.format;
  cout << "\ndata: " << in_event->xclient.data.b << endl;
}

//   Displays information about event received by program.
// Not currently used, but left in for future use.

int id_event(XEvent event_to_check) {

  switch(event_to_check.type) {
  case KeyPress:
    cout << "KeyPress";
    break;
  case KeyRelease:
    cout << "KeyRelease";
    break;
  case ButtonPress:
    cout << "ButtonPress";
    break;
  case ButtonRelease:
    cout << "ButtonRelease";
    break;
  case MotionNotify:
    cout << "MotionNotify";
    break;
  case EnterNotify:
    cout << "EnterNotify";
    break;
  case LeaveNotify:
    cout << "LeaveNotify";
    break;
  case FocusIn:
    cout << "FocusIn";
    break;
  case FocusOut:
    cout << "FocusOut";
    break;
  case KeymapNotify:
    cout << "KeymapNotify";
    break;
  case Expose:
    cout << "Expose";
    break;
  case GraphicsExpose:
    cout << "GraphicsExpose";
    break;
  case NoExpose:
```

```
    cout << "NoExpose";
    break;
  case VisibilityNotify:
    cout << "VisibilityNotify";
    break;
  case CreateNotify:
    cout << "CreateNotify";
    break;
  case DestroyNotify:
    cout << "DestroyNotify";
    break;
  case UnmapNotify:
    cout << "UnmapNotify";
    break;
  case MapNotify:
    cout << "MapNotify";
    break;
  case MapRequest:
    cout << "MapRequest";
    break;
  case ReparentNotify:
    cout << "ReparentNotify";
    break;
  case ConfigureNotify:
    cout << "ConfigureNotify";
    break;
  case ConfigureRequest:
    cout << "ConfigureRequest";
    break;
  case GravityNotify:
    cout << "GravityNotify";
    break;
  case ResizeRequest:
    cout << "ResizeRequest";
    break;
  case CirculateNotify:
    cout << "CirculateNotify";
    break;
  case CirculateRequest:
    cout << "CirculateRequest";
    break;
  case PropertyNotify:
    cout << "PropertyNotify";
    break;
  case SelectionClear:
    cout << "SelectionClear";
    break;
  case SelectionRequest:
    cout << "SelectionRequest";
    break;
  case SelectionNotify:
    cout << "SelectionNotify";
    break;
  case ColormapNotify:
```

```
    cout << "ColormapNotify";
    break;
  case ClientMessage:
    cout << "ClientMessage";
    break;
  case MappingNotify:
    cout << "MappingNotify";
    break;
  default:
    cout << event_to_check.type;
  }
  cout << " Event, send_event ";
  if(event_to_check.xany.send_event)
    cout << "TRUE";
  else
    cout << "FALSE";
  cout << ", Target Window: " << hex << event_to_check.xany.window << endl;
  return event_to_check.type;
}

//   Changes the value displayed in the title box below the
// draw window.

void set_title() {

  if(graphic_list.name() != NULL)
    XtVaSetValues(graph_title, XmNvalue, graphic_list.name(), NULL);
}

//   translations provides the mappings for the keyboard
// mapping table that allow the drawing canvas to capture
// mouse and keyboard events.

int main(unsigned int argc, char **argv) {
  Widget toplevel, main_w, menubar, rowcol, scrolled_win,
         op_button, term_button,
         stream_button, property_button, select_button;
  XtAppContext app;
  Pixmap op_button_pixmap, term_button_pixmap,
    stream_button_pixmap, property_button_pixmap,
    select_button_pixmap;
  XGCValues gcv1, gcv2, gcv3;
  Screen *screen_ptr;
  XtActionsRec actions;
  String translations =
    "<Btn1Down>:   draw(down)\n\
     <Btn1Up>:     draw(up)  \n\
     <Btn1Motion>: draw(motion)\n\
     <Key>:        draw(key)\n\
     <Key>Tab:    draw(tab)";
  unsigned long gc_mask;
  Window root_window, toplevel_window;

  toplevel = XtVaAppInitialize(&app, "Graph_editor", options,
```

```
                                XtNumber(options), &argc, argv,
                                NULL, NULL);

display_ptr = XtDisplay(toplevel);
XtGetApplicationResources(toplevel, (XtPointer) &Resrcs,
                          resources, XtNumber(resources),
                          NULL, 0);
screen_ptr = XtScreen(toplevel);
initialize_color_table(screen_ptr);
root_window = RootWindowOfScreen(screen_ptr);
gcv1.foreground = BlackPixelOfScreen(screen_ptr);
gcv1.background = WhitePixelOfScreen(screen_ptr);
gcv2.foreground = BlackPixelOfScreen(screen_ptr);
gcv2.background = WhitePixelOfScreen(screen_ptr);
gcv3.foreground = WhitePixelOfScreen(screen_ptr);
gcv3.background = WhitePixelOfScreen(screen_ptr);
gc_mask = GCForeground | GCBackground;
std_graphics_context = XCreateGC(display_ptr,
                    root_window, gc_mask, &gcv1);
dotted_context = XCreateGC(display_ptr,
                    root_window, gc_mask, &gcv2);
erase_context = XCreateGC(display_ptr, root_window, gc_mask,
                          &gcv3);
XSetLineAttributes(display_ptr, dotted_context, 1,
                   LineOnOffDash, CapButt, JoinMiter);
XSetFunction(display_ptr, dotted_context, GXxor);

if(Resrcs.viewer == False) {  //  editor mode, not view mode
  main_w = XtVaCreateManagedWidget("main_w", xmFormWidgetClass,
                                   toplevel, NULL);
  build_menu_bar(main_w, menubar);
  XtManageChild(menubar);
  rowcol =
    XtVaCreateManagedWidget("rowcol", xmRowColumnWidgetClass,
                            main_w,
                            XmNnumColumns, 1,
                            NULL);

  make_buttons(rowcol, op_button, term_button, stream_button,
               property_button, select_button,
               op_button_pixmap, term_button_pixmap,
               stream_button_pixmap, property_button_pixmap,
               select_button_pixmap, display_ptr, screen_ptr);

  XtAddCallback(op_button, XmNactivateCallback, op_button_cb,
                NULL);
  XtAddCallback(term_button, XmNactivateCallback,
                term_button_cb, NULL);
  XtAddCallback(stream_button, XmNactivateCallback,
                stream_button_cb, NULL);
  XtAddCallback(select_button, XmNactivateCallback,
                select_button_cb, NULL);
  XtAddCallback(property_button, XmNactivateCallback,
                property_button_cb, NULL);
```

```
    scrolled_win =
      XtVaCreateManagedWidget("scrolled_win",
                          xmScrolledWindowWidgetClass,
                          main_w,
//                          XmNwidth, 1200,
//                          XmNheight, 750,
                          XmNscrollingPolicy, XmAUTOMATIC,
                          XmNscrollBarDisplayPolicy, XmAS_NEEDED,
                          NULL);
    actions.string = "draw";
    actions.proc = draw;
    XtAppAddActions(app, &actions, 1);
    graph_title =
      XtVaCreateManagedWidget("graph_title", xmTextWidgetClass,
                              main_w,
                              XmNvalue, "",
                              NULL);
    tool_indicator = XtVaCreateManagedWidget("tool_indicator",
                                        xmTextWidgetClass,
                                        main_w,
                                        XmNvalue, "Select Tool",
                                        NULL);
    XtVaSetValues(toplevel, XmNtitle, "Graph Editor", NULL);
  }
  else {
    scrolled_win =
      XtVaCreateManagedWidget("scrolled_win",
                          xmScrolledWindowWidgetClass, toplevel,
//                          XmNwidth, 600,
//                          XmNheight, 400,
                          XmNscrollingPolicy, XmAUTOMATIC,
                          XmNscrollBarDisplayPolicy, XmAS_NEEDED,
                          NULL);
    XtVaSetValues(toplevel, XmNtitle, "Graph Viewer", NULL);
  }

  drawing_a =
    XtVaCreateManagedWidget("drawing_a",
                         xmDrawingAreaWidgetClass, scrolled_win,
                         XmNunitType, Xm1000TH_INCHES,
                         XmNwidth, 11000,
                         XmNheight, 8500,
                         XmNresizePolicy, XmNONE,
                         NULL);
  XtAddCallback(drawing_a, XmNexposeCallback, redraw, NULL);

  XtVaSetValues(drawing_a, XmNunitType, XmPIXELS, NULL);
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);
  XtVaGetValues(drawing_a, XmNwidth, &width, XmNheight, &height,
                NULL);
  drawing_area_pixmap = XCreatePixmap(display_ptr,
                          root_window, width, height,
                          DefaultDepthOfScreen(screen_ptr));
  XFillRectangle(display_ptr, drawing_area_pixmap,
```

```
                erase_context, 0, 0, width, height);

if(Resrcs.viewer == False) {
  XtVaSetValues(drawing_a,  XmNtranslations,
                XtParseTranslationTable(translations));
  XtVaSetValues(menubar, XmNtopAttachment, XmATTACH_FORM,
                XmNrightAttachment, XmATTACH_FORM,
                XmNleftAttachment, XmATTACH_WIDGET,
                XmNleftWidget, rowcol,
                XmNbottomAttachment, XmATTACH_NONE,
                NULL);

  XtVaSetValues(rowcol, XmNtopAttachment, XmATTACH_FORM,
                XmNrightAttachment, XmATTACH_NONE,
                XmNleftAttachment, XmATTACH_FORM,
                XmNbottomAttachment, XmATTACH_WIDGET,
                XmNbottomWidget, tool_indicator,
                NULL);
  XtVaSetValues(scrolled_win, XmNtopAttachment,
                XmATTACH_WIDGET,
                XmNtopWidget, menubar,
                XmNrightAttachment, XmATTACH_FORM,
                XmNleftAttachment, XmATTACH_WIDGET,
                XmNleftWidget, rowcol,
                XmNbottomAttachment, XmATTACH_WIDGET,
                XmNbottomWidget, graph_title,
                NULL);
  XtVaSetValues(graph_title, XmNtopAttachment, XmATTACH_NONE,
                XmNrightAttachment, XmATTACH_FORM,
                XmNleftAttachment, XmATTACH_WIDGET,
                XmNleftWidget, tool_indicator,
                XmNbottomAttachment, XmATTACH_FORM,
                NULL);
  XtVaSetValues(tool_indicator, XmNtopAttachment,
                XmATTACH_NONE,
                XmNrightAttachment, XmATTACH_NONE,
                XmNleftAttachment, XmATTACH_FORM,
                XmNbottomAttachment, XmATTACH_FORM,
                NULL);
}
XtRealizeWidget(toplevel);
draw_window = XtWindow(drawing_a);
graphic_list.set_draw_environ(display_ptr,
                              std_graphics_context,
                              erase_context, dotted_context,
                              draw_window,
                              &drawing_area_pixmap,
                              color_table,
                              width, height);
graphic_list.set_error_tgt(drawing_a);
graphic_list.build_from_disk();
if(Resrcs.viewer == False)
  set_title();
graphic_list.draw();
```

```
  XmProcessTraversal(drawing_a, XmTRAVERSE_CURRENT);

  if(Resrcs.viewer != False) {
    toplevel_window = XtWindowOfObject(toplevel);
    Atom display_id_atom = XInternAtom(display_ptr, "WINDOW_ID",
                                       False);
    XChangeProperty(display_ptr, root_window, display_id_atom,
                    XA_WINDOW, 32, PropModeReplace,
                    (unsigned char *) &toplevel_window, 1);
    XtAddEventHandler(toplevel, NoEventMask, TRUE, event_handler,
                      NULL);
  }
  printf("\n"); //flushes the event queue
  XFlush(display_ptr);
  XtAppMainLoop(app);
  return 0;
}
```

```
/* ************************************************************

Name:           graph_object.h
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 11 Sep 92
Remarks:   Specification for the GraphObject class.

          The GraphObject is the base class for the main graph
        objects displayed in the graph editor. It is not
        designed to be directly instantiated, so most of its
        methods are virtual.

          There are a number of static class variables used
        to store graphic information necessary for the
        descendents to draw themselves.

************************************************************* */
#ifndef graph_object_h
#define graph_object_h 1

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include "ge_defs.h"
#include "font_table.h"

class GraphObjectList;

class GraphObject {
protected:
  static Display *_display_ptr;
  static  GC _graphics_context, _erase_context, _dotted_context;
  static Window _draw_window;
  static Pixmap *_drawing_area_pixmap;
  static unsigned long _color_table[MAXCOLORS];
  static FontTable *_font_table;
  static int _default_font;
  static Widget _error_tgt;

  GraphObject *_next_ptr;

public:

  static Dimension window_width, window_height;
  static BOOLEAN valid_num_string(char *num);
  static void set_draw_environ(Display *display_ptr,
                               GC graphics_context,
                               GC erase_context,
                               GC dotted_context,
                               Window draw_window,
                               Pixmap *drawing_area_pixmap,
                               unsigned long color_table[],
                               Dimension width,
```

```
                                Dimension height);
static void font_init(Display *display_ptr);
static void set_default_font(int font_id)
                                        {_default_font = font_id;}
static int font_text_width(int font_id, char *in_string);
static int font_text_height(int font_id);
static char *font_name(int font_id)
                    {return _font_table->font_name(font_id);}
static void set_font(GC graphics_context, int font_id);
static void set_error_tgt(Widget widget) {_error_tgt = widget;}
static void error_box(char *error_message);

GraphObject();
~GraphObject() {delete _next_ptr;}
void link(GraphObject &next_object) {_next_ptr = &next_object;}
GraphObject* next() {return _next_ptr;}
virtual GE_STATUS build_from_property();
virtual GE_STATUS build_from_disk(FILE *) {return FAILED;}
virtual GE_STATUS write_to_property() {return FAILED;}
virtual GE_STATUS write_to_disk(FILE *) {return FAILED;}
virtual void draw(DRAW_STYLE) {}
virtual void select() {}
virtual void unselect() {}
virtual void erase() {}
virtual BOOLEAN hit(int, int) {return FALSE;}
virtual CLASS_DEF is_a() {return GRAPHOBJECT;}
virtual void set_object_ptrs(GraphObjectList *) {}
virtual int id() {return 0;}
virtual char *name() {return "";}
virtual void set_location(int, int) {}
virtual void set_deleted() {}
virtual void delete_notify(CLASS_DEF, int) {}
virtual void replace_name(char *) {}
virtual void set_constraint(int) {}
virtual void set_modified() {}
virtual BOOLEAN text_selected() {return FALSE;}
Display *display_ptr() {return _display_ptr;}
GC graphics_context() {return _graphics_context;}
Window draw_window() {return _draw_window;}
Pixmap *drawing_area_pixmap() {return _drawing_area_pixmap;}
virtual void move_notify(CLASS_DEF, int) {}
virtual void move_handle(int, int) {}
virtual BOOLEAN hit_handle(int, int) {return FALSE;}
virtual void set_color(COLOR) {}
virtual void set_object_font(int) {};
virtual void erase_text() {}
virtual void move_text(int, int) {}
virtual void draw_text(DRAW_STYLE) {}
virtual BOOLEAN is_deleted() {return FALSE;}
virtual void undelete_notify(CLASS_DEF, int) {}
virtual void reset_handles_drawn_state() {}
virtual int text_width() {return 0;}
virtual int text_height() {return 0;}
virtual void text_locate(int, int) {}
```

```
  virtual int constraint() {return 0;}
};

#endif;
```

```
/* ************************************************************

Name:           graph_object.C
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 11 Sep 92
Remarks:   Implementation for the GraphObject class.

           The GraphObject is the base class for the main graph
         objects displayed in the graph editor. It is not
         designed to be directly instantiated, so most of its
         methods are virtual.

Credits:    Portions of code are adapted from the following:

          Barakati, Naba, X Window System Programming, SAMS,
            1991.

          Heller, Dan, Motif Programming Manual, O'Reilly and
            Associates, 1991.

          Johnson, Eric, and Reichard, Kevin, X Window
            Applications Programming, MIS Press, 1989.

          Young, Douglas, Object Oriented Programming With C++
            and OSF/Motif, Prentice-Hall, 1992.

************************************************************ */
#include <Xm/MessageB.h>
#include <stream.h>
#include "graph_object.h"

//   Initializers for the static class variables.

int GraphObject::_default_font = 0;
Widget GraphObject::_error_tgt = NULL;

//   Determines whether the string represents a valid number.

BOOLEAN GraphObject::valid_num_string(char *num) {
  int index, num_length;

  if(num != NULL) {
    num_length = strlen(num);
    if(num_length > 0) {
      for(index = 0; index < num_length; index++) {
        if((num[index] != '\n') &&
           (num[index] != '-') &&
           ((num[index] < '0') || (num[index] > '9')))
          return FALSE;
      }
      return TRUE;
    }
```

```
  }
  return FALSE;
}

//   Displays an error message in a standard Motif error
// dialog box.

void GraphObject::error_box(char *error_message) {
  static Widget error_dialog = NULL;
  Arg arg[1];
  XmString t;

  if(_error_tgt != NULL) {
    if(error_dialog == NULL) {
      error_dialog = XmCreateMessageDialog(_error_tgt, "error",
                                           arg, 0);
      XtVaSetValues(XtParent(error_dialog),
                    XtNtitle, "Error",
                    NULL);
      XtUnmanageChild(XmMessageBoxGetChild(error_dialog,
                                      XmDIALOG_HELP_BUTTON));
    }
    t = XmStringCreateSimple(error_message);
    XtVaSetValues(error_dialog,
                  XmNmessageString, t,
                  NULL);
    XmStringFree(t);
    XtManageChild(error_dialog);
  }
  else
    cout <<
      "Error Target Widget must be set by calling module.\n"
        << endl;
}

GraphObject::GraphObject() {
  _next_ptr = NULL;
}

//   Sets up the drawing environment for the graph objects.

void GraphObject::set_draw_environ(Display *display_ptr,
                                   GC graphics_context,
                                   GC erase_context,
                                   GC dotted_context,
                                   Window draw_window,
                                   Pixmap *drawing_area_pixmap,
                                   unsigned long color_table[],
                                   Dimension width,
                                   Dimension height) {
  int i;

  _display_ptr = display_ptr;
  _graphics_context = graphics_context;
```

```
  _erase_context = erase_context;
  _dotted_context = dotted_context;
  _draw_window = draw_window;
  _drawing_area_pixmap = drawing_area_pixmap;
  for(i = 0; i <= MAXCOLORS; i++)
    _color_table[i] = color_table[i];
  GraphObject::window_width = width;
  GraphObject::window_height = height;
}

//   Initializes the font table.

void GraphObject::font_init(Display *display_ptr) {

  _font_table = new FontTable();
  _font_table->init(display_ptr);
}

//  Returns the width in pixels of the requested font.

int GraphObject::font_text_width(int font_id, char *in_string) {

  return _font_table->width(font_id, in_string);
}

//   Returns the height in pixels of the requested font.

int GraphObject::font_text_height(int font_id) {

  return _font_table->height(font_id);
}

//   Sets the requested font as the drawing font.

void GraphObject::set_font(GC graphics_context, int font_id) {

  XSetFont(_display_ptr, graphics_context,
           _font_table->font_id(font_id));
}

GE_STATUS GraphObject::build_from_property() {return FAILED;}
```

```
/* ************************************************************

Name:           graph_object_list.h
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 11 Sep 92
Remarks:   Specification for the GraphObjectList class.

          A GraphObjectList is a linked list of graphic
        objects, implemented as GraphObjects. Currently,
        a GraphObject may be either an OperatorObject or
        a StreamObject.

          Although not currently implemented this way, most
        of the functionality necessary to draw the graph title
        on the drawing canvas is present.

************************************************************ */
#ifndef graph_object_list_h
#define graph_object_list_h 1

#include <X11/Intrinsic.h>
#include "ge_defs.h"
#include "graph_object.h"

class GraphObject;

class GraphObjectList {
  protected:
    GraphObject *_head_ptr;
    int _last_op_id, _last_stream_id, _name_font, _name_x,
      _name_y, _name_width, _name_height;
    Display *_display_ptr;
    Window _draw_window;
    GC _graphics_context, _erase_context, _dotted_context;
    Pixmap *_drawing_area_pixmap;
    Dimension _width, _height;
    BOOLEAN _title_selected;
    char *_name_ptr;
    Widget _error_tgt;

    void error_box(char *error_message);
    static BOOLEAN valid_num_string(char *num);
    void set_text_dimensions();
    void set_name_location();
    void draw_handles(GC draw_context, int x1, int y1, int x2,
                      int y2);

  public:
    GraphObjectList();
    ~GraphObjectList() {delete _head_ptr;delete _name_ptr;}
    GE_STATUS build_from_property();
    GE_STATUS build_from_disk();
    GE_STATUS write_to_property();
```

```
    GE_STATUS write_to_disk(char *filename, int return_info);
    void draw();
    void erase();
    void draw_text(GC draw_context);
    void erase_text();
    void move_text(int x, int y);
    GraphObject* hit(int x, int y);
    CLASS_DEF is_a() {return GRAPHOBJECTLIST;}
    GraphObject* target_object(CLASS_DEF object_type, int id);
    void delete_notify(CLASS_DEF class_type, int deleted_obj_id);
    int request_id(CLASS_DEF class_type);
    void add(GraphObject *new_object_ptr);
    void set_draw_environ(Display *display_ptr,
                          GC graphics_context, GC erase_context,
                          GC dotted_context, Window draw_window,
                          Pixmap *drawing_area_pixmap,
                          unsigned long color_table[],
                          Dimension width, Dimension height);
    void get_del_op_list(char *del_op_str[], int del_op_id[],
                         int &num_del_ops);
    void set_undeleted(CLASS_DEF class_type, int id);
    void set_default_font(int font_id);
    void move_notify(CLASS_DEF object_type, int object_id);
    char *font_name(int font_id)
                       {return GraphObject::font_name(font_id);}
    BOOLEAN hit_text(int x, int y);
    void set_text_font(int font_id);
    void select_title();
    void unselect_title();
    char *name() {return _name_ptr;}
    void set_error_tgt(Widget widget);
};

#endif
```

```
/* ************************************************************

Name:           graph_object_list.C
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 11 Sep 92
Remarks:    Implementation of a GraphObjectList.

            A GraphObjectList is a linked list of graphic
          objects, implemented as GraphObjects. Currently,
          a GraphObject may be either an OperatorObject or
          a StreamObject.

            Although not currently implemented this way, most
          of the functionality necessary to draw the graph title
          on the drawing canvas is present.


Credits:     Portions of code are adapted from the following:

           Barakati, Naba, X Window System Programming, SAMS,
             1991.

           Heller, Dan, Motif Programming Manual, O'Reilly and
             Associates, 1991.

           Johnson, Eric, and Reichard, Kevin, X Window
             Applications Programming, MIS Press, 1989.

           Young, Douglas, Object Oriented Programming With C++
             and OSF/Motif, Prentice-Hall, 1992.


************************************************************* */
#include <stdio.h>
#include <string.h>
#include <stream.h>
#include <Xm/MessageB.h>
#include "graph_object_list.h"
#include "operator_object.h"
#include "stream_object.h"

//   Determines whether the input string represents a valid
// number.

BOOLEAN GraphObjectList::valid_num_string(char *num) {
  int index, num_length;

  if(num != NULL) {
    num_length = strlen(num);
    if(num_length > 0) {
      for(index = 0; index < num_length; index++) {
        if((num[index] != '\n') &&
           (num[index] != '-') &&
```

```
            ((num[index] < '0') || (num[index] > '9')))
          return FALSE;
      }
      return TRUE;
    }
  }
  return FALSE;
}

//   Displays a standard Motif error dialog box with the
// given error message.

void GraphObjectList::error_box(char *error_message) {
  static Widget error_dialog = NULL;
  Arg arg[1];
  XmString t;

  if(_error_tgt != NULL) {
    if(error_dialog == NULL) {
      error_dialog = XmCreateMessageDialog(_error_tgt, "error",
                                           arg, 0);
      XtVaSetValues(XtParent(error_dialog),
                    XtNtitle, "Error",
                    NULL);
      XtUnmanageChild(XmMessageBoxGetChild(error_dialog,
                                           XmDIALOG_HELP_BUTTON));
    }
    t = XmStringCreateSimple(error_message);
    XtVaSetValues(error_dialog,
                  XmNmessageString, t,
                  NULL);
    XmStringFree(t);
    XtManageChild(error_dialog);
  }
  else
    cout <<
      "Error Target Widget must be set by calling module.\n"
        << endl;
}

GraphObjectList::GraphObjectList() {

  _head_ptr = NULL;
  _name_ptr = NULL;
  _last_op_id = 0;
  _last_stream_id = 0;
  _title_selected = FALSE;
  _error_tgt = NULL;
}

GE_STATUS GraphObjectList::build_from_property() {return FAILED;}

//   Builds the GraphObjectList from disk.
```

```
GE_STATUS GraphObjectList::build_from_disk() {
  FILE *infile;
  char buffer[INPUT_LINE_SIZE + 1], *error_str_ptr = NULL;
  GraphObject *temp_object_ptr, *current_ptr;
  GE_STATUS status = SUCCEEDED;
  int last_char;

  delete _head_ptr;
  _head_ptr = NULL;
  if((infile = fopen("gedatatransfile.txt", "rt")) != NULL) {
    delete _name_ptr;
    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(strcmp(buffer, "GE93DATAFILE\n") == 0) {
      fgets(buffer, INPUT_LINE_SIZE, infile);
      last_char = strlen(buffer) - 1;
      if(buffer[last_char] == '\n')
        buffer[last_char] = 0;
      _name_ptr = strdup(buffer);
      fgets(buffer, INPUT_LINE_SIZE, infile);
      if(valid_num_string(buffer)) {
        _name_font = atoi(buffer);
        if(_name_font > MAXFONTS)
          error_str_ptr = strdup("Name Font Too Large");
      }
      else
        error_str_ptr = strdup("Corrupted header information");

      if(error_str_ptr == NULL) {
        fgets(buffer, INPUT_LINE_SIZE, infile);
        if(valid_num_string(buffer))
          _name_x = atoi(buffer);
        else
          error_str_ptr = strdup("Corrupted name_x");
      }

      if(error_str_ptr == NULL) {
        fgets(buffer, INPUT_LINE_SIZE, infile);
        if(valid_num_string(buffer))
          _name_y = atoi(buffer);
        else
          error_str_ptr = strdup("Corrupted name_y");
      }
      if(error_str_ptr == NULL) {
        set_text_dimensions();
        if(_name_x == NULL)
          set_name_location();
      }

      if(error_str_ptr == NULL) {
          //Clears the "OPERATORS" keyword from the input buffer
        fgets(buffer, INPUT_LINE_SIZE, infile);
        while(status == SUCCEEDED) {
          temp_object_ptr = new OperatorObject;
          status = temp_object_ptr->build_from_disk(infile);
```

```
        if(status == SUCCEEDED) {
                              //else all operators are read in
          if(temp_object_ptr->id() > _last_op_id)
            _last_op_id = temp_object_ptr->id();
          if(_head_ptr == NULL) {
            _head_ptr = temp_object_ptr;
            current_ptr = _head_ptr;
          }
          else {
            current_ptr->link(*temp_object_ptr);
            current_ptr = current_ptr->next();
          }
        }
      }
      delete temp_object_ptr;

//   Operators are read in; time for the streams

      if(status == ENDED) {
        status = SUCCEEDED;
        while((!feof(infile)) && (status == SUCCEEDED)) {
          temp_object_ptr = new StreamObject;
          status = temp_object_ptr->build_from_disk(infile);
          if(status == SUCCEEDED) {
            if(temp_object_ptr->id() > _last_stream_id)
              _last_stream_id = temp_object_ptr->id();
            temp_object_ptr->set_object_ptrs(this);
            if(_head_ptr == NULL) {
              _head_ptr = temp_object_ptr;
              current_ptr = _head_ptr;
            }
            else {
              current_ptr->link(*temp_object_ptr);
              current_ptr = current_ptr->next();
            }
          }
          else
            delete temp_object_ptr;
        }
      }
    }
    else {
      error_box(error_str_ptr);
      status = FAILED;
      free(error_str_ptr);
    }
  }
  else {
    error_box("Data file format is questionable.");
    fclose(infile);
    status = FAILED;
  }
  fclose(infile);
}
```

```
  else {
    error_box("Data transfer file not found.");
    status = FAILED;
  }
  return status;
}

GE_STATUS GraphObjectList::write_to_property() {return FAILED;}

//   Writes the GraphObjectList to disk

GE_STATUS GraphObjectList::write_to_disk(char *filename,
                                         int return_info) {
  FILE *outfile;
  GraphObject *temp_object_ptr = _head_ptr;
  GE_STATUS status = SUCCEEDED;
  char buffer[INPUT_LINE_SIZE + 1];

  outfile = fopen(filename, "wt");
  if(outfile != NULL) {
    fprintf(outfile, "GE93DATAFILE\n");
    fprintf(outfile, "%s\n", _name_ptr);
    sprintf(buffer, "%d\n%d\n%d\n", _name_font, _name_x,
            _name_y);
    fprintf(outfile, buffer);
    fprintf(outfile, "OPERATORS\n");
    while((temp_object_ptr != NULL) && (status == SUCCEEDED)) {
      if(temp_object_ptr->is_a() == OPERATOROBJECT)
        status = temp_object_ptr->write_to_disk(outfile);
      temp_object_ptr = temp_object_ptr->next();
    }
    if(status == SUCCEEDED) {
      fprintf(outfile, "STREAMS\n");
      temp_object_ptr = _head_ptr;
      while((temp_object_ptr != NULL) && (status == SUCCEEDED)) {
        if(temp_object_ptr->is_a() == STREAMOBJECT)
          status = temp_object_ptr->write_to_disk(outfile);
        temp_object_ptr = temp_object_ptr->next();
      }
      if(status == SUCCEEDED) {
        fprintf(outfile, "ENDDATA\n");
        sprintf(buffer, "%d\n", return_info);
        fprintf(outfile, buffer);
        if(return_info > 0)
          fprintf(outfile,
           (target_object(OPERATOROBJECT, return_info))->name());
      }
    }
    fclose(outfile);
  }
  else
    status = FAILED;
  if(status == FAILED)
    error_box("Disk Write Failed");
```

```
  return status;
}

//   Draws the GraphObjectList to the drawing canvas.

void GraphObjectList::draw() {
  GraphObject *temp_object_ptr = _head_ptr;

  XFillRectangle(_display_ptr, _draw_window, _erase_context, 0,
                 0, _width, _height);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 _erase_context, 0, 0, _width, _height);

//   Streams are drawn first to prevent the operators' handles
// from overwriting the arrowheads.  Handles are drawn in xor
// mode so they will erase properly.

//   Since handles are drawn in xor mode, it's important to
// track whether they've been drawn or not.  Redrawing the
// handles unintentionally erases them.

  while(temp_object_ptr != NULL) {
    if(temp_object_ptr->is_a() == STREAMOBJECT) {
      temp_object_ptr->reset_handles_drawn_state();
      temp_object_ptr->draw(SOLID);
    }
    temp_object_ptr = temp_object_ptr->next();
  }
  temp_object_ptr = _head_ptr;
  while(temp_object_ptr != NULL) {
    if(temp_object_ptr->is_a() == OPERATOROBJECT) {
      temp_object_ptr->reset_handles_drawn_state();
      temp_object_ptr->draw(SOLID);
    }
    temp_object_ptr = temp_object_ptr->next();
  }
}

void GraphObjectList::erase() {}

//   If the given coordinates are located inside one of the
// graph objects or their text strings, the function returns
// a pointer to the object.

GraphObject* GraphObjectList::hit(int x, int y) {
  GraphObject *temp_object_ptr = _head_ptr;

  while(temp_object_ptr != NULL) {
    if(temp_object_ptr->hit(x, y))
      return temp_object_ptr;
    else
      temp_object_ptr = temp_object_ptr->next();
  }
  return (GraphObject *) NULL;
```

```
}

//   Returns a pointer to the requested object.

GraphObject* GraphObjectList::target_object(
                                  CLASS_DEF object_type, int id) {
  GraphObject* temp_object_ptr = _head_ptr;
  while(temp_object_ptr != NULL) {
    if((temp_object_ptr->is_a() == object_type) &&
       (temp_object_ptr->id() == id)) {
      return temp_object_ptr;
    }
    temp_object_ptr = temp_object_ptr->next();
  }
  error_box("Requested operator id not found");
  return NULL;
}

//   Notifies all the objects that one of them has been
// deleted, so that they may take appropriate actions.

void GraphObjectList::delete_notify(CLASS_DEF class_type,
                                    int deleted_obj_id) {
  GraphObject *temp_object_ptr = _head_ptr;

  while(temp_object_ptr != NULL) {
    temp_object_ptr->delete_notify(class_type, deleted_obj_id);
    temp_object_ptr = temp_object_ptr->next();
  }
}

//   When a new object is added to the list, it needs a
// unique identifier. The GraphObjectList tracks the
// highest stream and operator identifiers and issues the
// next higher on request.

int GraphObjectList::request_id(CLASS_DEF class_type) {

  if(class_type == OPERATOROBJECT) {
    _last_op_id++;
    return _last_op_id;
  }
  else {
    _last_stream_id++;
    return _last_stream_id;
  }
}

//   Adds a new GraphObject to the list.

void GraphObjectList::add(GraphObject *new_object_ptr) {
  GraphObject *temp_object_ptr = _head_ptr;

  if(_head_ptr == NULL)
```

```
    _head_ptr = new_object_ptr;
  else
    if(_head_ptr->next() == NULL)
      _head_ptr->link(*new_object_ptr);
    else {
      while(temp_object_ptr->next() != NULL)
        temp_object_ptr = temp_object_ptr->next();
      temp_object_ptr->link(*new_object_ptr);
    }
}

//   Sets the necessary drawing variables, and performs the
// same operation for the GraphObject class.

void GraphObjectList::set_draw_environ(Display *display_ptr,
                GC graphics_context, GC erase_context,
                GC dotted_context, Window draw_window,
                Pixmap *drawing_area_pixmap,
                unsigned long color_table[], Dimension width,
                Dimension height) {

  _display_ptr = display_ptr;
  _graphics_context = graphics_context;
  _erase_context = erase_context;
  _dotted_context = dotted_context;
  _draw_window = draw_window;
  _drawing_area_pixmap = drawing_area_pixmap;
  _width = width;
  _height = height;
  GraphObject::set_draw_environ(_display_ptr, _graphics_context,
              _erase_context, _dotted_context, _draw_window,
              _drawing_area_pixmap, color_table, width, height);
  GraphObject::font_init(_display_ptr);
}

//   Notifies the objects that one of them has been moved.

void GraphObjectList::move_notify(CLASS_DEF object_type,
                                  int object_id) {
  GraphObject *temp_object_ptr = _head_ptr;

  while(temp_object_ptr != NULL) {
    temp_object_ptr->move_notify(object_type, object_id);
    temp_object_ptr = temp_object_ptr->next();
  }
}

//   Sets the default font.

void GraphObjectList::set_default_font(int font_id) {

GraphObject::set_default_font(font_id);
}
```

```
//   Sets the dimensions for the title string so it can be
// drawn on the canvas. Left in for future use.

void GraphObjectList::set_text_dimensions() {

  if(_name_ptr == NULL) {
    _name_width = 0;
    _name_height = 0;
  }
  else {
    _name_width = GraphObject::font_text_width(_name_font, _name_ptr);
    _name_height = GraphObject::font_text_height(_name_font);
  }
}

//   Sets a default name location. Left in for future use.

void GraphObjectList::set_name_location() {

  _name_y = 100;
  _name_x = (_width - _name_width) / 2;
}

//   Draws the title string on the canvas. Left in for
// future use.

void GraphObjectList::draw_text(GC draw_context) {

  if(_name_ptr != NULL) {
    GraphObject::set_font(draw_context, _name_font);
    XDrawString(_display_ptr, _draw_window, draw_context, _name_x,
                _name_y, _name_ptr, strlen(_name_ptr));
      XDrawString(_display_ptr, *_drawing_area_pixmap, draw_context,
                  _name_x, _name_y, _name_ptr, strlen(_name_ptr));
      if(_title_selected)
        draw_handles(draw_context, _name_x - HANDLESIZE,
                     _name_y - _name_height - HANDLESIZE,
                     _name_x + _name_width + HANDLESIZE,
                     _name_y + HANDLESIZE);
  }
}

//   Erases the title string. Left in for future use.

void GraphObjectList::erase_text() {

  draw_text(_erase_context);
}

//   Moves the title string.  Left in for future use.

void GraphObjectList::move_text(int x, int y) {

  if(_title_selected) {
```

```
    _name_x += x;
    _name_y += y;
  }
}

//   Returns TRUE if the mouse clicked on the title string,
// FALSE otherwise.

BOOLEAN GraphObjectList::hit_text(int x, int y) {

  if(((x >= _name_x) && (x <= (_name_x + _name_width))) &&
     (y >= _name_y - _name_height) && (y <= _name_y)) {
    _title_selected = TRUE;
    return TRUE;
  }
  else
    return FALSE;
}

//   Draws a handle on the graph.

void GraphObjectList::draw_handles(GC draw_context, int x1,
                                   int y1, int x2, int y2) {

  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y2 - HANDLESIZE, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y1,
                 HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y2 - HANDLESIZE,
                 HANDLESIZE, HANDLESIZE);
}

//   Sets the name font to the desired value. Left in for
// future use.

void GraphObjectList::set_text_font(int font_id) {

  _name_font = font_id;
  set_text_dimensions();
}
```

```
//   Unselects the title.

void GraphObjectList::unselect_title() {

  erase_text();
  _title_selected = FALSE;
  draw_text(_graphics_context);
}

//   Selects the title.

void GraphObjectList::select_title() {

  erase_text();
  _title_selected = TRUE;
  draw_text(_graphics_context);
}

//   Makes a list of deleted operators.

void GraphObjectList::get_del_op_list(char *del_op_str[],
                                      int del_op_id[],
                                      int &num_del_ops) {
  GraphObject *temp_obj_ptr = _head_ptr;
  int index = 0;

  while((temp_obj_ptr != NULL) && (index < MAXDELETEDOPS)) {
    if((temp_obj_ptr->is_a() == OPERATOROBJECT) &&
       (temp_obj_ptr->is_deleted())) {
         del_op_str[index] = strdup(temp_obj_ptr->name());
         del_op_id[index] = temp_obj_ptr->id();
         index++;
     }
     temp_obj_ptr = temp_obj_ptr->next();
  }
  num_del_ops = index;
}

//   Notifies the objects that the given object has been
// undeleted.

void GraphObjectList::set_undeleted(CLASS_DEF class_type,
                                    int id) {
  GraphObject *temp_obj_ptr = _head_ptr;

  while(temp_obj_ptr != NULL) {
    temp_obj_ptr->undelete_notify(class_type, id);
    temp_obj_ptr = temp_obj_ptr->next();
  }
}

//   Sets the widget used to display the error message box.
```

```
void GraphObjectList::set_error_tgt(Widget widget) {

  _error_tgt = widget;
  GraphObject::set_error_tgt(widget);
  SplineObject::set_error_tgt(widget);
  FontTable::set_error_tgt(widget);
}
```

```
/* ************************************************************

Name:           operator_object.h
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 17 Sep 92
Remarks:   This is the specification for the OperatorObject
         class.

           The OperatorObject class is the graphical
         representation of a PSDL operator.  It has three
         physical forms: circular for an atomic operator,
         two concentric circles for a non-atomic operator,
         and rectangular, for terminators.  A terminator
         simulates interaction with objects outside the system.

************************************************************* */
#ifndef operator_object_h
#define operator_object_h 1

#include <stdio.h>
#include <X11/Xlib.h>
#include "ge_defs.h"
#include "graph_object.h"

class OperatorObject : public GraphObject {
protected:
  char *_name_ptr, *_met_string_ptr;
  int _id, _met, _x, _y, _radius, _color, _name_font, _name_x,
    _name_y, _name_width, _name_height, _met_font, _met_x,
    _met_y, _met_width, _met_height, _handle_selected;
  BOOLEAN _is_deleted, _is_new, _is_modified, _is_composite,
  _is_terminator, _is_selected, _name_selected, _met_selected,
  _op_handles_drawn, _name_handles_drawn, _met_handles_drawn;

  void set_default_met_location();
  void set_default_name_location();
  void set_text_dimensions();
  void draw_handles(GC draw_context, int x1, int y1, int x2,
                    int y2);
  void set_object_font(int font_id);
  void reset_handles_drawn_state();
public:
  OperatorObject();
  OperatorObject(char *in_name_ptr, int in_id, int in_met,
                 int in_x, int in_y, int in_radius, int in_color,
                 BOOLEAN in_is_new, BOOLEAN in_is_composite,
                 BOOLEAN in_is_terminator);
  ~OperatorObject();
  GE_STATUS build_from_property();
  GE_STATUS build_from_disk(FILE *infile);
  GE_STATUS write_to_property();
  GE_STATUS write_to_disk(FILE *outfile);
  void draw(DRAW_STYLE style);
```

```
  void draw_text(DRAW_STYLE style);
  void set_default_text_location();
  void select();
  void unselect();
  void erase();
  void erase_text();
  void move_text(int x, int y);
  BOOLEAN hit(int x, int y);
  CLASS_DEF is_a() {return OPERATOROBJECT;}
  int id() {return _id;}
  char *name() {return _name_ptr;}
  int x() {return _x;}
  int y() {return _y;}
  int radius() {return _radius;}
  BOOLEAN is_composite() {return _is_composite;}
  BOOLEAN is_terminator() {return _is_terminator;}
  XYPAIR center();
  XYPAIR intercept(int x, int y);
  void set_location(int x, int y);
  void move(int x, int y);
  void set_deleted() {_is_deleted = TRUE;}
  void undelete_notify(CLASS_DEF class_type, int deleted_obj_id);
  BOOLEAN is_deleted() {return _is_deleted;}
  void delete_notify(CLASS_DEF class_type, int deleted_obj_id);
  void replace_name(char *new_name);
  void set_constraint(int constraint);
  void set_modified() {_is_modified = TRUE;}
  void move_notify(CLASS_DEF , int ) {}
  BOOLEAN hit_handle(int x, int y);
  void move_handle(int x, int y);
  void set_radius(int radius) {_radius = radius;}
  void set_color(COLOR color) {_color = color;}
  BOOLEAN text_selected()
                    {return (_name_selected || _met_selected);}
  int text_width();
  int text_height();
  void text_locate(int x, int y);
  void write_block(GC draw_context, int x, int y,
                   char *instring, int block_height);
  int handle_selected() {return _handle_selected;}
  void set_handle_selected(int handle);
  int constraint() {return _met;}
};

#endif;
```

```
/* ************************************************************

Name:           operator_object.C
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 17 Sep 92
Remarks:   This is the implementation of the OperatorObject
         class.

           The OperatorObject class is the graphical
         representation of a PSDL operator.  It has three
         physical forms: circular for an atomic operator,
         two concentric circles for a non-atomic operator,
         and rectangular, for terminators.  A terminator
         simulates interaction with objects outside the system.


Credits:    Portions of code are adapted from the following:

          Barakati, Naba, X Window System Programming, SAMS,
            1991.

          Heller, Dan, Motif Programming Manual, O'Reilly and
            Associates, 1991.

          Johnson, Eric, and Reichard, Kevin, X Window
            Applications Programming, MIS Press, 1989.

          Young, Douglas, Object Oriented Programming With C++
            and OSF/Motif, Prentice-Hall, 1992.


************************************************************ */
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <stream.h>
#include "operator_object.h"

#define COMPOSITE_SPACE 5
#define NUM_STRING_LENGTH 15
#define NONE 0
#define UPPERLEFT 1
#define UPPERRIGHT 2
#define LOWERRIGHT 3
#define LOWERLEFT 4
#define MINRADIUS 5

//   Draws handles around the given coordinates.  Drawn in
// exclusive-or mode to simplify erasure.

void OperatorObject::draw_handles(GC draw_context, int x1,
                                  int y1, int x2, int y2) {
```

```
  XSetFunction(_display_ptr, draw_context, GXxor);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y2 - HANDLESIZE, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y1, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y2 - HANDLESIZE,
                 HANDLESIZE, HANDLESIZE);
  XSetFunction(_display_ptr, draw_context, GXcopy);
}

//   Determines the dimensions of the text blocks.  _name_font
// and _met_font must be set before calling.  Text strings
// are automatically broken at underscores, except for
// underscores in the first and last character of the string.

void OperatorObject::set_text_dimensions() {
  int i, end_prev_index = -1, widest_string = 0, num_breaks = 0,
    widest_start_index, temp_width, temp_height, str_len;
  char temp_str[INPUT_LINE_SIZE];

  if(_name_ptr == NULL) {
    _name_width = 0;
    _name_height = 0;
  }
  else {
    str_len = strlen(_name_ptr);
    for(i = 0; i < str_len; i++)
      if(_name_ptr[i] == '_') {
        if((i != 0) && (i != (str_len - 1))) {
          num_breaks++;
          temp_width = i - end_prev_index;
          if(temp_width > widest_string) {
            widest_string = temp_width;
            widest_start_index = end_prev_index + 1;
          }
          end_prev_index = i;
        }
      }
    if(end_prev_index < (str_len - 1)) {
      temp_width = str_len - 1 - end_prev_index;
```

```
      if(temp_width > widest_string) {
        widest_string = temp_width;
        widest_start_index = end_prev_index + 1;
      }
    }
    if(widest_string == 0) {
      widest_start_index = 0;
      widest_string = str_len;
    }
    strncpy(temp_str, &(_name_ptr[widest_start_index]),
            widest_string);
    temp_str[widest_string] = NULL;
    _name_width = font_text_width(_name_font, temp_str);
    temp_height = font_text_height(_name_font);
    _name_height = temp_height * (num_breaks + 1);
  }
  if(_met != NULL_VALUE) {
    _met_width = font_text_width(_met_font, _met_string_ptr);
    _met_height = font_text_height(_met_font);
  }
  else {
    _met_width = 0;
    _met_height = 0;
  }
}

OperatorObject::OperatorObject() : GraphObject() {

  _name_ptr = NULL;
  _name_font = _default_font;
  _name_x = NULL_VALUE;
  _name_y = NULL_VALUE;
  _met_string_ptr = NULL;
  _met_font = _default_font;
  _met_x = NULL_VALUE;
  _met_y = NULL_VALUE;
  _is_selected = FALSE;
  _is_deleted = FALSE;
  _is_modified = FALSE;
  _handle_selected = NONE;
  _name_selected = FALSE;
  _met_selected = FALSE;
  reset_handles_drawn_state();
  _name_width = 0;
  _name_height = 0;
  _met_width = 0;
  _met_height = 0;
}

//   MET values less than zero are considered to represent
// microsecond values.

OperatorObject::OperatorObject(char *in_name_ptr, int in_id,
                               int in_met, int in_x, int in_y,
```

```
                                int in_radius, int in_color,
                                BOOLEAN in_is_new,
                                BOOLEAN in_is_composite,
                                BOOLEAN in_is_terminator) {
  char buffer[INPUT_LINE_SIZE];

  _name_ptr = strdup(in_name_ptr);
  _id = in_id;
  _met = in_met;
  if(_met == NULL_VALUE)
    _met_string_ptr = NULL;
  else {
    if(_met < 0)
      sprintf(buffer, "%d us", -_met);
    else
      sprintf(buffer, "%d ms", _met);
    _met_string_ptr = strdup(buffer);
  }
  _x = in_x;
  _y = in_y;
  _radius = in_radius;
  _color = in_color;
  _is_new = in_is_new;
  _is_deleted = FALSE;
  _is_modified = FALSE;
  _is_composite = in_is_composite;
  _is_terminator = in_is_terminator;
  _is_selected = FALSE;
  _name_selected = FALSE;
  _met_selected = FALSE;
  _handle_selected = NONE;
  _name_font = _default_font;
  _met_font = _default_font;
  set_text_dimensions();
  set_default_text_location();
  reset_handles_drawn_state();
}

OperatorObject::~OperatorObject() {

  delete _name_ptr;
  delete _met_string_ptr;
}

//   Determines a default location for the MET string.

void OperatorObject::set_default_met_location() {

  if(_is_terminator)
    _met_x = _x + (int) ((float) _radius * 1.5)
      - (_met_width / 2);
  else
    _met_x = _x + _radius
      - (_met_width / 2);
```

```
  _met_y = _y;
}

//   Determines a default location for the name string.

void OperatorObject::set_default_name_location() {

  if(_is_terminator)
    _name_x = _x + (int) ((float) _radius * 1.5)
      - (_name_width / 2);
  else
    _name_x = _x + _radius
      - (_name_width / 2);
  _name_y = _y + _radius + (_name_height / 2);
}

//   Convenience routine for setting both locations at once.

void OperatorObject::set_default_text_location() {

  set_default_name_location();
  set_default_met_location();
}

GE_STATUS OperatorObject::build_from_property() {return FAILED;}

//   Builds an operator from a disk file.

GE_STATUS OperatorObject::build_from_disk(FILE *infile) {
  char buffer[INPUT_LINE_SIZE + 1], *error_str_ptr = NULL;
  int last_char;
  GE_STATUS status = SUCCEEDED;

  fgets(buffer, INPUT_LINE_SIZE, infile);
  last_char = strlen(buffer) - 1;
  if(buffer[last_char] == '\n') // Strips off trailing newline
    buffer[last_char] = 0;
  if(strcmp(buffer, "STREAMS") == 0)
    return ENDED;
  else {
    _name_ptr = strdup(buffer);

    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer)) {
      _name_font = atoi(buffer);
      if(_name_font > MAXFONTS)
        error_str_ptr = strdup("Name Font Too Large");
    }
    else
      error_str_ptr = strdup("Corrupted Name Font");

    if(error_str_ptr == NULL) {
      fgets(buffer, INPUT_LINE_SIZE, infile);
      if(valid_num_string(buffer))
```

```
    _name_x = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted name_x");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _name_y = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted name_y");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _id = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted id");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _met = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted met");
}


if(error_str_ptr == NULL) {
  if(_met == NULL_VALUE)
    _met_string_ptr = NULL;
  else {
    if(_met < 0)
      sprintf(buffer, "%d us", -_met);
    else
      sprintf(buffer, "%d ms", _met);
    _met_string_ptr = strdup(buffer);
  }
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer)) {
    _met_font = atoi(buffer);
    if(_met_font > MAXFONTS)
      error_str_ptr = strdup("Met Font Too Large");
  }
  else
    error_str_ptr = strdup("Corrupted Met Font");
}

if(error_str_ptr == NULL)
```

```
  set_text_dimensions();

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _met_x = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted met_x");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _met_y = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted met_y");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _x = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted x");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _y = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted y");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _radius = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted radius");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer)) {
    _color = atoi(buffer);
    if(_color > MAXCOLORS)
      error_str_ptr = strdup("Color Too Large");
  }
  else
    error_str_ptr = strdup("Corrupted Color");
}

if(error_str_ptr == NULL) {
```

```
  if(strcmp("TRUE\n",
            fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
    _is_deleted = TRUE;
  else
    if(strcmp(buffer, "FALSE\n") == 0)
      _is_deleted = FALSE;
    else
      error_str_ptr = strdup("Corrupted is_deleted");
}

if(error_str_ptr == NULL) {
  if(strcmp("TRUE\n",
            fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
      //Change this if original _is_new should be saved.
    _is_new = FALSE;
    else
      if(strcmp(buffer, "FALSE\n") == 0)
        _is_new = FALSE;
      else
        error_str_ptr = strdup("Corrupted is_new");
}

if(error_str_ptr == NULL) {
  if(strcmp("TRUE\n",
            fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
    _is_modified = TRUE;
  else
    if(strcmp(buffer, "FALSE\n") == 0)
      _is_modified = FALSE;
    else
      error_str_ptr = strdup("Corrupted is_modified");
}

if(error_str_ptr == NULL) {
  if(strcmp("TRUE\n",
            fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
    _is_composite = TRUE;
  else
    if(strcmp(buffer, "FALSE\n") == 0)
      _is_composite = FALSE;
    else
      error_str_ptr = strdup("Corrupted is_composite");
}

if(error_str_ptr == NULL) {
  if(strcmp("TRUE\n",
            fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
    _is_terminator = TRUE;
  else
    if(strcmp(buffer, "FALSE\n") == 0)
      _is_terminator = FALSE;
    else
      error_str_ptr = strdup("Corrupted is_terminator");
}
```

```
    if(error_str_ptr == NULL) {
      if(_name_x == NULL_VALUE)
        set_default_name_location();
      if(_met_x == NULL_VALUE)
        set_default_met_location();
    }

    if(error_str_ptr != NULL) {
      sprintf(buffer, "operator %s: %s", _name_ptr,
              error_str_ptr);
      error_box(buffer);
      status = FAILED;
      free(error_str_ptr);
    }

    if(ferror(infile)) {
      sprintf(buffer,
        "Unix reported an error building operator %s",
          _name_ptr);
      error_box(buffer);
      clearerr(infile);
      status = FAILED;
    }
  }
  return status;
}

GE_STATUS OperatorObject::write_to_property() {return FAILED;}

//   Writes the operator's attributes to disk.

GE_STATUS OperatorObject::write_to_disk(FILE *outfile) {
  char buffer[INPUT_LINE_SIZE + 1];

  fprintf(outfile, "%s\n", _name_ptr);
  sprintf(buffer, "%d\n%d\n%d", _name_font, _name_x, _name_y);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", _id);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", _met);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d\n%d\n%d", _met_font, _met_x, _met_y);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", _x);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", _y);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", _radius);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", _color);
  fprintf(outfile, "%s\n", buffer);
  if(_is_deleted)
    fprintf(outfile, "TRUE\n");
```

```
  else
    fprintf(outfile, "FALSE\n");
  if(_is_new)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(_is_modified)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(_is_composite)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(_is_terminator)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(ferror(outfile)) {
    clearerr(outfile);
    return FAILED;
  }
  else
    return SUCCEEDED;
}

//   Writes a text block to the drawing canvas.  Text strings
// are broken at underscores, except when in the first and
// last character position.

void OperatorObject::write_block(GC draw_context, int x, int y,
    char *instring, int block_height) {
  int i, str_len, num_lines = 1, string_width, start_index = 0,
  yinc, block_index = 0;
  char *block_text[MAXTEXTLINES], temp_string[INPUT_LINE_SIZE];

  if(instring != NULL) {
    str_len = strlen(instring);
    if(str_len != 0) {
      for(i = 0; i < str_len; i++)
        if(instring[i] == '_') {
          if((i != 0) && (i != (str_len - 1))) {
            string_width = i - start_index + 1;
            strncpy(temp_string, &(instring[start_index]),
                    string_width);
            temp_string[string_width] = NULL;
            block_text[num_lines - 1] = strdup(temp_string);
            start_index = i + 1;
            num_lines++;
          }
        }
      if(start_index < str_len) {
        string_width = i - start_index + 1;
        strncpy(temp_string, &(instring[start_index]),
```

```
                string_width);
        temp_string[string_width] = NULL;
        block_text[num_lines - 1] = strdup(temp_string);
      }
      else
        num_lines--;
      yinc = block_height / num_lines;
      for(i = 0; i < num_lines; i++) {
        str_len = strlen(block_text[i]);
        XDrawString(_display_ptr, _draw_window, draw_context, x,
                    y - (yinc * (num_lines - i - 1)),
                    block_text[i], str_len);
        XDrawString(_display_ptr, *_drawing_area_pixmap,
                    draw_context, x,
                    y - (yinc * (num_lines - i - 1)),
                    block_text[i], str_len);
        free(block_text[i]);
      }
    }
  }
}

//   Writes text on the drawing canvas, including handles if
// appropriate.  Since handles are drawn in exclusive-or mode,
// it's important to make sure that they aren't redrawn
// unless they've been erased or they need to be erased.

void OperatorObject::draw_text(DRAW_STYLE style) {
  GC draw_context;

  if((style == SOLID) || (style == DOTTED))
    draw_context = _graphics_context;
  else
    if(style == ERASE)
      draw_context = _erase_context;
  set_font(draw_context, _name_font);
  write_block(draw_context, _name_x, _name_y, _name_ptr,
              _name_height);
  if((_name_selected) && (_name_handles_drawn == FALSE) &&
     (style == SOLID)) {
    draw_handles(_graphics_context, _name_x - HANDLESIZE,
                 _name_y - _name_height - HANDLESIZE,
                 _name_x + _name_width + HANDLESIZE,
                 _name_y + HANDLESIZE);
    _name_handles_drawn = TRUE;
  }
  else
    if((_name_selected) && (_name_handles_drawn == TRUE) &&
       (style == ERASE)) {
      draw_handles(_graphics_context, _name_x - HANDLESIZE,
                   _name_y - _name_height - HANDLESIZE,
                   _name_x + _name_width + HANDLESIZE,
                   _name_y + HANDLESIZE);
      _name_handles_drawn = FALSE;
```

```
    }
  if(_met != NULL_VALUE) {
    set_font(draw_context, _met_font);
    XDrawString(_display_ptr, _draw_window, draw_context,
                _met_x, _met_y, _met_string_ptr,
                strlen(_met_string_ptr));
    XDrawString(_display_ptr, *_drawing_area_pixmap,
                draw_context, _met_x, _met_y, _met_string_ptr,
                strlen(_met_string_ptr));
    if((_met_selected) && (_met_handles_drawn == FALSE) &&
       (style == SOLID)) {
      draw_handles(_graphics_context, _met_x - HANDLESIZE,
                   _met_y - _met_height - HANDLESIZE,
                   _met_x + _met_width + HANDLESIZE,
                   _met_y + HANDLESIZE);
      _met_handles_drawn = TRUE;
    }
    else
      if((_met_selected) && (_met_handles_drawn == TRUE) &&
         (style == ERASE)) {
        draw_handles(_graphics_context, _met_x - HANDLESIZE,
                     _met_y - _met_height - HANDLESIZE,
                     _met_x + _met_width + HANDLESIZE,
                     _met_y + HANDLESIZE);
        _met_handles_drawn = FALSE;
      }

  }
}

//   Draws the operator.

void OperatorObject::draw(DRAW_STYLE style) {
  GC draw_context;

  if(_is_deleted == FALSE) {
    if(style == SOLID)
      draw_context = _graphics_context;
    else
      if(style == DOTTED)
        draw_context = _dotted_context;
      else
        draw_context = _erase_context;
    if(_is_terminator) {
      if(style == SOLID) {
        XSetForeground(_display_ptr, draw_context,
                       _color_table[_color]);
      }
      if(style != DOTTED) {
        XFillRectangle(_display_ptr, _draw_window, draw_context,
                       _x, _y, _radius * 3, _radius * 2);
        XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                       draw_context, _x, _y, _radius * 3,
                       _radius * 2);
```

```
    if(style == ERASE)
      _op_handles_drawn = FALSE;
  }
  if(style == SOLID)
    XSetForeground(_display_ptr, draw_context, _color_table[BLACK]);
  XDrawRectangle(_display_ptr, _draw_window, draw_context,
                 _x, _y, _radius * 3, _radius * 2);
  XDrawRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, _x, _y, _radius * 3,
                 _radius * 2);

  if((_is_selected) && (_op_handles_drawn == FALSE) &&
     (style == SOLID)) {
    draw_handles(_graphics_context, _x, _y,
                 _x + (3 * _radius), _y + (2 * _radius));
    _op_handles_drawn = TRUE;
  }
}
else {
  if(_is_composite) {
    if(style == SOLID) {
      XSetForeground(_display_ptr, draw_context,
                     _color_table[_color]);
    }
    if(style != DOTTED) {
      XFillArc(_display_ptr, _draw_window, draw_context,
               _x + COMPOSITE_SPACE, _y + COMPOSITE_SPACE,
               (_radius - COMPOSITE_SPACE) * 2,
               (_radius - COMPOSITE_SPACE) * 2,
               CIRCLE_BEGIN, FULL_CIRCLE);
      XFillArc(_display_ptr, *_drawing_area_pixmap,
               draw_context, _x + COMPOSITE_SPACE,
               _y + COMPOSITE_SPACE,
               (_radius - COMPOSITE_SPACE) * 2,
               (_radius - COMPOSITE_SPACE) * 2,
               CIRCLE_BEGIN, FULL_CIRCLE);
    }
    if(style == SOLID) {
      XSetForeground(_display_ptr, draw_context, _color_table[BLACK]);
    }
    XDrawArc(_display_ptr, _draw_window, draw_context, _x,
             _y, _radius * 2, _radius * 2, CIRCLE_BEGIN,
             FULL_CIRCLE);
    XDrawArc(_display_ptr, *_drawing_area_pixmap,
             draw_context, _x, _y, _radius * 2, _radius *2,
             CIRCLE_BEGIN, FULL_CIRCLE);
    XDrawArc(_display_ptr, _draw_window, draw_context,
             _x + COMPOSITE_SPACE, _y + COMPOSITE_SPACE,
             (_radius - COMPOSITE_SPACE) * 2,
             (_radius - COMPOSITE_SPACE) * 2, CIRCLE_BEGIN,
             FULL_CIRCLE);
    XDrawArc(_display_ptr, *_drawing_area_pixmap,
             draw_context, _x + COMPOSITE_SPACE,
             _y + COMPOSITE_SPACE,
```

```
                (_radius - COMPOSITE_SPACE) * 2,
                (_radius - COMPOSITE_SPACE) * 2, CIRCLE_BEGIN,
                FULL_CIRCLE);
      }
      else {
        if(style == SOLID) {
          XSetForeground(_display_ptr, draw_context,
          _color_table[_color]);
        }
        if(style != DOTTED) {
          XFillArc(_display_ptr, _draw_window, draw_context,
                   _x, _y, _radius * 2, _radius * 2,
                   CIRCLE_BEGIN, FULL_CIRCLE);
          XFillArc(_display_ptr, *_drawing_area_pixmap,
                   draw_context, _x, _y, _radius * 2,
                   _radius * 2, CIRCLE_BEGIN, FULL_CIRCLE);
        }
        if(style == SOLID) {
          XSetForeground(_display_ptr, draw_context, _color_table[BLACK]);
        }
        XDrawArc(_display_ptr, _draw_window, draw_context,
                 _x, _y, _radius * 2, _radius * 2, CIRCLE_BEGIN,
                 FULL_CIRCLE);
        XDrawArc(_display_ptr, *_drawing_area_pixmap,
                 draw_context, _x, _y, _radius * 2, _radius *2,
                 CIRCLE_BEGIN, FULL_CIRCLE);
      }
      if((_is_selected) && (_op_handles_drawn == FALSE) &&
         (style == SOLID)) {
        draw_handles(_graphics_context, _x, _y,
                     _x + (2 * _radius), _y + (2 * _radius));
        _op_handles_drawn = TRUE;
      }
      else
        if((_is_selected) && (_op_handles_drawn == TRUE) &&
           (style == ERASE)) {
          draw_handles(_graphics_context, _x, _y,
                       _x + (2 * _radius), _y + (2 * _radius));
          _op_handles_drawn = FALSE;
        }
    }
    draw_text(style);
  }
}

void OperatorObject::erase() {

  draw(ERASE);
}

void OperatorObject::erase_text() {

  draw_text(ERASE);
}
```

```
//   Returns true if the coordinates are located within either
// the operator or one of its text strings.

BOOLEAN OperatorObject::hit(int x, int y) {

  if(_is_deleted)
    return FALSE;
  else {
    if(_name_ptr != NULL) {
      if(strlen(_name_ptr) != 0)
        if(((x >= _name_x) && (x <= (_name_x + _name_width))) &&
           (y >= _name_y - _name_height) && (y <= _name_y)) {
          _name_selected = TRUE;
          return TRUE;
        }
    }
    if(_met_string_ptr != NULL) {
      if(strlen(_met_string_ptr) != 0)
        if(((x >= _met_x) && (x <= (_met_x + _met_width))) &&
           (y >= _met_y - _met_height) && (y <= _met_y)) {
          _met_selected = TRUE;
          return TRUE;
        }
    }
    if(_is_terminator) {
      if(((x >= _x) && (x <= (_x + (3 * _radius)))) &&
         (y >= _y) && (y <= (_y + (2 * _radius))))
        return TRUE;
      else
        return FALSE;
    }
    else {
      if(((x >= _x) && (x <= (_x + (2 * _radius)))) &&
         (y >= _y) && (y <= (_y + (2 * _radius))))
        return TRUE;
      else
        return FALSE;
    }
  }
}

//   Returns the coordinates of the center of the given
// operator.

XYPAIR OperatorObject::center() {
  XYPAIR temp_pair;

  if(_is_terminator) {
    temp_pair.x = _x + (int) ((float) _radius * 1.5);
    temp_pair.y = _y + _radius;
  }
  else {
    temp_pair.x = _x + _radius;
```

```
    temp_pair.y = _y + _radius;
  }
  return temp_pair;
}

//   Given the last coordinate, returns the point on the
// circumference of the operator where streams should begin
// or end.

XYPAIR OperatorObject::intercept(int x, int y) {
  int distance;
  float slope;
  XYPAIR temp_pair, obj_center;

  obj_center = center();
  if(_is_terminator) {
    slope = (float) (y - obj_center.y) /
                                    (float) (x - obj_center.x);
    if(fabs(slope) >= (2.0 / 3.0)) {
      if(y <= _y)
        temp_pair.y = _y;
      else
        temp_pair.y = _y + (_radius * 2);
      temp_pair.x = (int) ((float) (temp_pair.y - obj_center.y) /
                                        slope) +  obj_center.x;
    }
    else {
      if(x <= _x)
        temp_pair.x = _x;
      else
        temp_pair.x = _x + (_radius * 3);
      temp_pair.y = (int) ((float) (temp_pair.x - obj_center.x) *
                                         slope) + obj_center.y;
    }
  }
  else {
    distance = (int) sqrt(((x - obj_center.x) * (x - obj_center.x)) +
                        (y - obj_center.y) * (y - obj_center.y));
    temp_pair.x = obj_center.x + (int)
                   (((float) _radius/ (float) distance) *
                                    (float) (x - obj_center.x));
    temp_pair.y = obj_center.y + (int)
                    (((float) _radius/(float) distance) *
                      (float) (y - obj_center.y));
  }
  return temp_pair;
}

//   Moves the operator the given distance.

void OperatorObject::move(int x, int y) {

  _x += x;
  _y += y;
```

```
  if(_x < 0)
    _x = 0;
  if(_y < 0)
    _y = 0;
  _name_x += x;
  _name_y += y;
  _met_x += x;
  _met_y += y;
}

//   Relocates the operator to the given position.  x and y
// represent the center of the operator.

void OperatorObject::set_location(int x, int y) {
  int old_x = _x, old_y = _y;

  if(_is_terminator)
    _x = x - (int) ((float) _radius * 1.5);
  else
    _x = x - _radius;
  _y = y - _radius;
  if(_x < 0)
    _x = 0;
  if(_y < 0)
    _y = 0;
  if(_name_x == NULL_VALUE)
    set_default_name_location();
  else {
    _name_x += _x - old_x;
    _name_y += _y - old_y;
  }
  if(_met_x == NULL_VALUE)
    set_default_met_location();
  else {
    _met_x += _x - old_x;
    _met_y += _y - old_y;
  }
  reset_handles_drawn_state();
}

//   Included for symmetry.  Streams delete themselves when
// their operators are deleted, so this function is included
// for possible future use.

void OperatorObject::delete_notify(CLASS_DEF class_type,
                       int deleted_object_id) {}

void OperatorObject::replace_name(char *new_name) {

  delete _name_ptr;
  _name_ptr = strdup(new_name);
  set_text_dimensions();
  set_default_name_location();
}
```

```
void OperatorObject::unselect() {

  erase();
  _is_selected = FALSE;
  _name_selected = FALSE;
  _met_selected = FALSE;
  draw(SOLID);
}

void OperatorObject::select() {

  _is_selected = TRUE;
  draw(SOLID);
}

//   Returns TRUE if one of the handles is in the given
// location.

BOOLEAN OperatorObject::hit_handle(int x, int y) {

  if((_x <= x) && (x <= (_x + HANDLESIZE)) &&
                        (_y <= y) && (y <= (_y + HANDLESIZE))) {
    _handle_selected = UPPERLEFT;
    return TRUE;
  }
  if((_x <= x) && (x <= (_x + HANDLESIZE)) &&
     ((_y + 2 * _radius - HANDLESIZE)<= y) &&
     (y <= (_y + 2 * _radius))) {
    _handle_selected = LOWERLEFT;
    return TRUE;
  }
  if(_is_terminator) {
    if(((_x + 3 * _radius - HANDLESIZE) <= x) &&
       (x <= (_x + 3 * _radius)) &&
       (_y <= y) && (y <= (_y + HANDLESIZE))) {
      _handle_selected = UPPERRIGHT;
      return TRUE;
    }
    if(((_x + 3 * _radius - HANDLESIZE) <= x) &&
       (x <= (_x + 3 * _radius)) &&
       ((_y + 2 * _radius - HANDLESIZE) <= y) &&
        (y <= (_y + 2 * _radius))) {
      _handle_selected = LOWERRIGHT;
      return TRUE;
    }
  }
  else {
    if(((_x + (_radius * 2) - HANDLESIZE) <= x) &&
       (x <= (_x +  2 * _radius)) &&
       (_y <= y) && (y <= (_y + HANDLESIZE))) {
      _handle_selected = UPPERRIGHT;
      return TRUE;
    }
```

```
    if(((_x + 2 * _radius - HANDLESIZE) <= x) &&
       (x <= (_x + 2 * _radius)) &&
       ((_y + 2 * _radius - HANDLESIZE) <= y) &&
       (y <= (_y + 2 * _radius))) {
      _handle_selected = LOWERRIGHT;
      return TRUE;
    }
  }
  _handle_selected = NONE;
  return FALSE;
}

//   When the handle is dragged, the operator is resized.

void OperatorObject::move_handle(int x, int y) {
  int radius_change = y / 2;
//used to eliminate "floating" due to roundoff

  draw(DOTTED);
  if(_handle_selected == UPPERLEFT) {
    _radius -= radius_change;
    if(_radius >= MINRADIUS) {
      if(_is_terminator)
        _x +=  3 * radius_change;
      else
        _x +=  2 * radius_change;
      _y +=  2 * radius_change;
    }
    else
      _radius += radius_change;
  }
  else
  if(_handle_selected == UPPERRIGHT) {
    _radius -= radius_change;
    if(_radius >= MINRADIUS)
      _y +=  2 * radius_change;
    else
      _radius += radius_change;
  }
  else
    if(_handle_selected == LOWERRIGHT) {
      _radius += radius_change;
      if(_radius < MINRADIUS)
        _radius -= radius_change;
    }
    else
      if(_handle_selected == LOWERLEFT) {
        _radius += radius_change;
        if(_radius >= MINRADIUS) {
          if(_is_terminator)
            _x -=  3 * radius_change;
          else
            _x -=  2 * radius_change;
        }
```

```
        else
          _radius -= radius_change;
      }
//   Places the text in default location. May not be desired.
//  set_default_text_location();
  draw(DOTTED);
}

//   Changes the MET to that entered in the properties dialog
// box.

void OperatorObject::set_constraint(int constraint) {
  char buffer[INPUT_LINE_SIZE];
  int old_met = _met;

  _met = constraint;
  delete _met_string_ptr;
  if(_met == NULL_VALUE)
    _met_string_ptr = NULL;
  else {
    if(_met < 0)
      sprintf(buffer, "%d us", -_met);
    else
      sprintf(buffer, "%d ms", _met);
    _met_string_ptr = strdup(buffer);
  }
  set_text_dimensions();
  if(old_met == NULL_VALUE)
    set_default_met_location();
}

//   Changes the font of the selected text string.

void OperatorObject::set_object_font(int font_id) {

  if(_name_selected)
    _name_font = font_id;
  if(_met_selected)
    _met_font = font_id;
  set_text_dimensions();
}

//   Moves the text.

void OperatorObject::move_text(int x, int y) {

  erase_text();
  if(_name_selected) {
    _name_x += x;
    _name_y += y;
  }
  else
    if(_met_selected) {
      _met_x += x;
```

```
      _met_y += y;
    }
  draw_text(SOLID);
}

//   Handles undeleted operators.

void OperatorObject::undelete_notify(CLASS_DEF class_type,
                                     int deleted_obj_id) {

  if((class_type == OPERATOROBJECT) &&
      (deleted_obj_id == _id)) {
    _is_deleted = FALSE;
    _is_modified = TRUE;
    _is_selected = FALSE;
    _name_selected = FALSE;
    _met_selected = FALSE;
    _handle_selected = NONE;
  }
}

void OperatorObject::reset_handles_drawn_state() {

  _op_handles_drawn = FALSE;
  _name_handles_drawn = FALSE;
  _met_handles_drawn = FALSE;
}

int OperatorObject::text_height() {

  if(_name_selected)
    return _name_height;
  else
    if(_met_selected)
      return _met_height;
    else
      return 0;
}

int OperatorObject::text_width() {

  if(_name_selected)
    return _name_width;
  else
    if(_met_selected)
      return _met_width;
    else
      return 0;
}

//   Moves the text to the desired location.

void OperatorObject::text_locate(int x, int y) {
```

```
  if(_met_selected) {
    _met_x = x - _met_width / 2;
    _met_y = y + _met_height / 2;
  }
  else
    if(_name_selected) {
      _name_x = x - _name_width / 2;
      _name_y = y + _name_height / 2;
    }
}

void OperatorObject::set_handle_selected(int handle) {

  _handle_selected = handle;
  _is_selected = TRUE;
}
```

```
/* ************************************************************

Name:           resources.h
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 21 Sep 92
Remarks:   Establishes common defines for the different
          colors and fonts used in the graph_editor.

************************************************************* */
#ifndef RESOURCES_H
#define RESOURCES_H 1

#define MAXCOLORS 64
#define AQUAMARINE 1
#define BLACK 2
#define BLUE 3
#define BLUEVIOLET 4
#define BROWN 5
#define CADETBLUE 6
#define CORAL 7
#define CORNFLOWERBLUE 8
#define CYAN 9
#define DARKGREEN 10
#define DARKOLIVEGREEN 11
#define DARKORCHID 12
#define DARKSLATEBLUE 13
#define DARKSLATEGREY 14
#define DARKTURQUOISE 15
#define DIMGREY 16
#define FIREBRICK 17
#define FORESTGREEN 18
#define GOLD 19
#define GOLDENROD 20
#define GREY 21
#define GREEN 22
#define GREENYELLOW 23
#define INDIANRED 24
#define KHAKI 25
#define LIGHTBLUE 26
#define LIGHTGREY 27
#define LIGHTSTEELBLUE 28
#define LIMEGREEN 29
#define MAGENTA 30
#define MAROON 31
#define MEDIUMAQUAMARINE 32
#define MEDIUMBLUE 33
#define MEDIUMORCHID 34
#define MEDIUMSEAGREEN 35
#define MEDIUMSLATEBLUE 36
#define MEDIUMSPRINGGREEN 37
#define MEDIUMTURQUOISE 38
#define MEDIUMVIOLETRED 39
#define MIDNIGHTBLUE 40
```

```
#define NAVY 41
#define ORANGE 42
#define ORANGERED 43
#define ORCHID 44
#define PALEGREEN 45
#define PINK 46
#define PLUM 47
#define RED 48
#define SALMON 49
#define SEAGREEN 50
#define SIENNA 51
#define SKYBLUE 52
#define SLATEBLUE 53
#define SPRINGGREEN 54
#define STEELBLUE 55
#define TAN 56
#define THISTLE 57
#define TURQUIOSE 58
#define VIOLET 59
#define VIOLETRED 60
#define WHEAT 61
#define WHITE 62
#define YELLOW 63
#define YELLOWGREEN 64

#define MAXFONTS 6
#define COURIERBOLD10 1
#define COURIERBOLD12 2
#define COURIERBOLD14 3
#define COURIERMED10 4
#define COURIERMED12 5
#define COURIERMED14 6

#endif
```

```
/* ************************************************************

Name:           sde.c
Author:         Capt Robert M. Dixon
Program:        sde
Date Modified: 21 Sep 92
Remarks:   The graph_editor program is written to interface
         with the CAPS '93 syntax-directed editor.  Since this
         editor is not finished, sde was written to exercise
         the necessary functions that the real syntax-directed
         editor would use.

           sde uses the same in-memory data structures that
         the syntax-directed editor uses, and interfaces to
         the graph editor in the same way.

************************************************************* */
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "sde_ge.h"
#include "globals.h"
#include "resources.h"

#define INPUTLINESIZE 100

/*   These three functions are provided by ge_interface.c to
 provide the graph editor support.  The syntax-directed
 editor invokes a copy of the graph editor in viewer mode,
 and then calls refresh() to update its display.

   To edit the graph, edit() is called.
   To make the viewer go away, kill_viewer() is called.
*/

extern void refresh();
extern void edit();
extern void kill_viewer();


 /* If changed, change in graph_editor.C also. */
#define NULL_VALUE 0

/*   Builds an operator node with the given values. */

struct op_node* build_op_node(name, id, met, x, y, radius,
                              color, is_deleted, is_new,
                              is_modified, is_composite,
                              is_terminator)
char *name;
OP_ID id;
int met, x, y, radius, color;
BOOL is_deleted, is_new, is_modified, is_composite,
  is_terminator;
```

```
{
  struct op_node* temp_node =
            (struct op_node *) malloc(sizeof(struct op_node));
  temp_node->op =
              (struct op_str *) malloc(sizeof(struct op_str));
  temp_node->op->name = strdup(name);
  temp_node->op->name_font = NULL_VALUE;
  temp_node->op->name_x = NULL_VALUE;
  temp_node->op->name_y = NULL_VALUE;

  temp_node->op->id = id;
  if(met == NULL_VALUE)
    temp_node->op->met = NULL;
  else {
    temp_node->op->met = (int *) malloc(sizeof(int));
    *(temp_node->op->met) = met;
  }
  temp_node->op->met_font = NULL_VALUE;
  temp_node->op->met_x = NULL_VALUE;
  temp_node->op->met_y = NULL_VALUE;

  temp_node->op->x = x;
  temp_node->op->y = y;
  temp_node->op->radius = radius;
  temp_node->op->color = color;
  temp_node->op->is_deleted = is_deleted;
  temp_node->op->is_new = is_new;
  temp_node->op->is_modified = is_modified;
  temp_node->op->is_composite = (BOOL *) malloc(sizeof(BOOL));
  *(temp_node->op->is_composite) = is_composite;
  temp_node->op->is_terminator = is_terminator;
  temp_node->next = NULL;
  return temp_node;
}

/*   Displays the values in the given operator node.  */

void  display_op_nodes() {
  struct op_node *temp_node_ptr = the_operator_list;

  while(temp_node_ptr != NULL) {
    printf("Name: %s ", temp_node_ptr->op->name);
    printf("Id: %d ", temp_node_ptr->op->id);
    printf("Font: %d %d %d ", temp_node_ptr->op->name_font,
         temp_node_ptr->op->name_x, temp_node_ptr->op->name_y);
    if(temp_node_ptr->op->met == NULL)
      printf("Met: None ");
    else
      printf("Met: %d ", *(temp_node_ptr->op->met));

    printf("x: %d ", temp_node_ptr->op->x);
    printf("y: %d ", temp_node_ptr->op->y);
    printf("radius: %d ", temp_node_ptr->op->radius);
    printf("color: %d ", temp_node_ptr->op->color);
```

```
    if(temp_node_ptr->op->is_deleted)
      printf("     deleted ");
    else
      printf(" not_deleted ");
    if(temp_node_ptr->op->is_new)
      printf("     new ");
    else
      printf(" not_new ");
    if(temp_node_ptr->op->is_modified)
      printf("     modified ");
    else
      printf(" not_modified ");
    if(temp_node_ptr->op->is_composite == NULL)
      printf(
           "is_composite pointer is NULL. Possible problem.\n");
    else {
      if(*(temp_node_ptr->op->is_composite))
        printf("     composite ");
      else
        printf(" not_composite ");
    }
    if(temp_node_ptr->op->is_terminator)
      printf("     terminator \n");
    else
      printf(" not_terminator \n");
    temp_node_ptr = temp_node_ptr->next;
  }
}

/*   Displays the values in the given stream node.  */

void  display_st_nodes() {
  struct st_node *temp_node_ptr = the_stream_list;
  struct spline_node *temp_spline_ptr;

  while(temp_node_ptr != NULL) {
    printf("Name: %s ", temp_node_ptr->st->name);
    printf("Id: %d ", temp_node_ptr->st->id);
    printf("From: ");
    if(temp_node_ptr->st->from != NULL)
      printf(" %d ", temp_node_ptr->st->from->op->id);
    else
      printf("0");
    printf("To: ");
    if(temp_node_ptr->st->to != NULL)
      printf("%d ", temp_node_ptr->st->to->op->id);
    else
      printf("0");
    printf("{");
    temp_spline_ptr = temp_node_ptr->st->arc;
    while(temp_spline_ptr != NULL) {
      printf(" %d %d,", temp_spline_ptr->x, temp_spline_ptr->y);
      temp_spline_ptr = temp_spline_ptr->next;
    }
```

```
    printf("}");
    printf("Latency: %d ", temp_node_ptr->st->latency);
    if(temp_node_ptr->st->is_deleted)
      printf("     deleted ");
    else
      printf(" not_deleted ");
    if(temp_node_ptr->st->is_new)
      printf("     new ");
    else
      printf(" not_new ");
    if(temp_node_ptr->st->is_modified)
      printf("     modified ");
    else
      printf(" not_modified ");
    if(temp_node_ptr->st->is_state_variable)
      printf("     state_variable \n");
    else
      printf(" not_state_variable \n");
    temp_node_ptr = temp_node_ptr->next;
  }
}

/*   Builds a stream node with the given parameters.  */

struct st_node* build_st_node(name, id, from, to, arc, latency,
                              is_deleted, is_new, is_modified,
                              is_state_variable)
  char *name;
  ST_ID id;
  OPNodePTR from, to;
  SPLINE_PTR arc;
  int latency;
  BOOL is_deleted, is_new, is_modified, is_state_variable;
{
  struct st_node* temp_node =
             (struct st_node *) malloc(sizeof(struct st_node));

  temp_node->st =
               (struct st_str *) malloc(sizeof(struct st_str));
  temp_node->st->name = strdup(name);
  temp_node->st->name_font = NULL_VALUE;
  temp_node->st->name_x = NULL_VALUE;
  temp_node->st->name_y = NULL_VALUE;
  temp_node->st->id = id;
  temp_node->st->from = from;
  temp_node->st->to = to;
  temp_node->st->arc = arc;
  temp_node->st->latency = latency;
  temp_node->st->latency_font = NULL_VALUE;
  temp_node->st->latency_x = NULL_VALUE;
  temp_node->st->latency_y = NULL_VALUE;
  temp_node->st->is_deleted = is_deleted;
  temp_node->st->is_new = is_new;
  temp_node->st->is_modified = is_modified;
```

```
  temp_node->st->is_state_variable = is_state_variable;
  temp_node->next = NULL;
  return temp_node;
}

/*   Builds a spline node with the given values.  */

struct spline_node* new_spline_node(x, y)
int x;
int y;
{
  struct spline_node *new_node =
      (struct spline_node *) malloc(sizeof(struct spline_node));

  new_node->x = x;
  new_node->y = y;
  new_node->next = NULL;
  return new_node;
}

/*   Builds a data structure with sample data. An operator is
  commented out for debugging purposes, but may be included
  in again by removing the comments.
*/

void initialize_data() {
  struct op_node *current_op_node;
  OPNodePTR op1_ptr, op2_ptr, op3_ptr;
  struct st_node *current_st_node;
  struct spline_node *spline1, *spline2, *spline3;

  prototype = (HeadPtr) malloc(sizeof(HEADER_NODE));
  prototype->next = NULL;
  current_graph = prototype;
  the_operator_list = build_op_node("OP1", 1, 0, 100, 100, 20,
                                    RED, FALSE, FALSE, FALSE,
                                    FALSE, FALSE);
  current_op_node = the_operator_list;
  prototype->operator_list = the_operator_list;
  current_op_node->next = build_op_node("OP2", 2, 500, 300, 100,
                                        20, BLUE, FALSE, FALSE,
                                        FALSE, TRUE, FALSE);
/*  current_op_node = current_op_node->next;
  current_op_node->next = build_op_node("OP3", 3, 0, 150, 150,
                                        20, YELLOW, FALSE, FALSE,
                                        FALSE, FALSE, TRUE);
*/
  op1_ptr = the_operator_list;
  op2_ptr = the_operator_list->next;
/*  op3_ptr = the_operator_list->next->next;
*/
  spline1 = new_spline_node(170, 50);
  spline1->next = new_spline_node(240, 50);
  spline2 = new_spline_node(300, 200);
```

```
  spline2->next = new_spline_node(200, 300);
  spline3 = new_spline_node(160, 250);
  spline3->next = new_spline_node(150, 200);
  the_stream_list = build_st_node("FLOW1", 1, op1_ptr, op2_ptr,
                                  spline1, 250, FALSE, FALSE,
                                  FALSE, TRUE);
  prototype->stream_list = the_stream_list;
  prototype->name = strdup("Sample Graph");
  prototype->name_font = 0;
  prototype->name_x = NULL_VALUE;
  prototype->name_y = NULL_VALUE;
  current_st_node = the_stream_list;
  current_st_node->next = build_st_node("FLOW2", 2, op2_ptr,
                                        NULL, spline2, 0, FALSE,
                                        FALSE, FALSE, FALSE);
  current_st_node = current_st_node->next;
  current_st_node->next = build_st_node("FLOW3", 3, NULL,
                                        op1_ptr, spline3, 0,
                                        FALSE, FALSE, FALSE,
                                        FALSE);
}

/*  Turns off the _is_new flags in the operator and stream lists.
  This prevents them from being added into the sde's data
  structure again the next time the graph editor is called.
*/

void make_old() {
  struct op_node *temp_op_ptr = the_operator_list;
  struct st_node *temp_st_ptr = the_stream_list;

  while(temp_op_ptr != NULL) {
    temp_op_ptr->op->is_new = FALSE;
    temp_op_ptr = temp_op_ptr->next;
  }
  while(temp_st_ptr != NULL) {
    temp_st_ptr->st->is_new = FALSE;
    temp_st_ptr = temp_st_ptr->next;
  }
}

/*   The commented out code is sometimes necessary for
  debugging.  The pdb debugger goes into an endless loop when
  it encounters a scanf() call, so the choice value must be
  hard-coded to use the debugger.
*/

main() {
int choice = 5;
FILE *tool_file;
char buffer[INPUTLINESIZE + 1];

  initialize_data();
```

```
/*  the_operator_list = NULL;
  the_stream_list = NULL;
*/
  if((tool_file = fopen("tool_location.txt",
                        "rt")) == NULL)
    printf("Tool location file not found.\n");
  else {
    fscanf(tool_file, "%s", buffer);
    while(strcmp(buffer, "graph_viewer:") != 0)
      fscanf(tool_file, "%s", buffer);
    fgets(buffer, INPUTLINESIZE, tool_file);
    fclose(tool_file);
    system(buffer);
    while(choice != 4) {
      printf("  1.  Graph Viewer\n");
      printf("  2.  Graph Editor\n");
      printf("  3.  Kill Viewer\n");
      printf("  4.  Quit\n\n");
      printf("choice: %d\n", choice);
      scanf("%d", &choice);

/*    choice = 2;
*/    switch(choice) {
      case 1:
        make_old();
        refresh();
        break;
      case 2:
        make_old();
#ifdef GE_DEBUG
        display_op_nodes();
        display_st_nodes();
#endif
        edit();
        if(level_change_direction == UP)
          printf("Change up\n");
        else
          if(level_change_direction == SAME)
            printf("No level change.\n");
          else
            printf("Change level: %s\n", goto_child);

#ifdef GE_DEBUG

printf("============================================================\n");
        display_op_nodes();
        display_st_nodes();
#endif
        break;
      case 3:
        kill_viewer();
        break;
      case 4:
        break;
```

```
      default:
        printf("Please choose 1, 2, 3, or 4.\n");
      }
    }
  }
}
```

```
/* ************************************************************

Name:           sde_ge.h
Author:         Capt Robert M. Dixon
Program:        sde
Date Modified: 12 Sep 92
Remarks:   Provides the common declarations for the
         syntax-directed editor and the interface routines.

           Originally created by Professor Valdis Berzins and
         modified by Professor Fernando Naveda and Capt Robert
         M. Dixon.


************************************************************ */
#ifndef SDE_GE_H
#define SDE_GE_H 1

typedef int BOOL;         /* 0 means false, 1 means true */
typedef int OP_ID;        /* unique identifier numbers for
                             OPERATORs, 1 .. # ops */

typedef struct op_str {
  char *name;
  int
        name_font,
        name_x,
        name_y;
  OP_ID id;         /* cannot be changed after creation */
  int
        *met,
        met_font,
        met_x,
        met_y,
        x,
        y,
        radius,
        color;         /* met in units of ms */
  BOOL
        is_deleted,
        is_new,        /* is_new is set by the GE and cleared by the SDE */
        *is_composite,
        is_terminator,
        is_modified;
}OP_NODE, *OPERATOR;

/*typedef struct op_str* OPERATOR;*/

typedef struct op_node {
  OPERATOR op;
  struct op_node *next;
}OP_HEAD, *OPNodePTR;

/*typedef struct op_node* OPERATOR_LIST;*/
```

```
/*-------------------------------------------------------------*/
typedef struct spline_node {
  int
        x,
        y;
  struct spline_node*
        next;
}SPLINE_NODE, *SPLINE_PTR;

/*typedef struct spline_node* SPLINE;*/
/*-------------------------------------------------------------*/
typedef int ST_ID;       /* unique identifier numbers for
                            streams, 1 .. # streams */

typedef struct st_str{
  char *name;
  int
        name_font,
        name_x,
        name_y;
        ST_ID id;          /* cannot be changed after creation */
  OPNodePTR
        from,
        to;
  SPLINE_PTR
        arc;               /* null pointer if the arc is a
                              straight line */
  int
        latency,           /* latency in units of ms */
        latency_font,
        latency_x,
        latency_y;
  BOOL
        is_deleted,
        is_new,            /* is_new is set by the GE and
                              cleared by the SDE */
        is_state_variable,
        is_modified;
}ST_NODE, *STREAM;

/*typedef struct st_str* STREAM;*/

typedef struct st_node {
  STREAM st;
  struct st_node
        *next;
}ST_HEAD, *ST_PTR;

/*typedef struct st_node* STREAM_LIST;*/
/*-------------------------------------------------------------*/
typedef struct Header_Node{
  char
        *name;
  int
```

```
        name_font,
        name_x,
        name_y,
        op_id_no;
  int
        met;
  BOOL
        marked_for_delete,
        is_composite;
  ST_PTR
        stream_list;
  OPNodePTR
        operator_list;
  struct Header_Node
        *next;
}HEADER_NODE, *HeadPtr;

#define UP 1
#define SAME 0
#define DOWN -1
#define TRUE 1
#define FALSE 0

#endif
```

```
/* ************************************************************

Name:           spline_object.h
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 11 Sep 92
Remarks:   Specification for the SplineObject class.

          The SplineObject is used to create the curved lines
        used in the graph_editor's splines.  It stores a
        linked list of control points which it uses to draw
        itself, point by point.

          In order to correctly terminate in the appropriate
        locations, it adds special control points I call
        'shadow points' at the ends of the lines.  Since
        b-splines don't normally end at the first and last
        control points, the extra shadow points make the
        curve stop in the right place.

************************************************************* */
#ifndef spline_object_h
#define spline_object_h 1

#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <Xm/MessageB.h>
#include "ge_defs.h"

class OperatorObject;
class StreamObject;

class _spline_node {
  friend SplineObject;
protected:
  int _x, _y;
  _spline_node *_next_ptr;
public:
  _spline_node() {_next_ptr = NULL;}
  _spline_node(int x, int y) {_x = x; _y = y; _next_ptr = NULL;}
  _spline_node(XYPAIR inpair)
    {_x = inpair.x; _y = inpair.y; _next_ptr = NULL;}
  ~_spline_node() {delete _next_ptr;}
};

class SplineObject {
protected:
  static Widget _error_tgt;

  _spline_node *_head_ptr;
  BOOLEAN _shadow_pts_set, _handles_drawn;
  _spline_node *_iter;
  int _handle_selected;
```

```
  static valid_num_string(char *num);

  void draw_arrowhead(GC graphics_context, StreamObject *parent,
                      _spline_node *endpoint);
  static void error_box(char *error_message);
  int num_points(_spline_node *p1, _spline_node *p2,
                 _spline_node *p3, _spline_node *p4);
public:
  static void set_error_tgt(Widget widget) {_error_tgt = widget;}

  SplineObject();
  ~SplineObject() {delete _head_ptr;}
  GE_STATUS build_from_disk(FILE *infile);
  GE_STATUS write_to_disk(FILE *outfile, EXTERN_STATUS status);
  CLASS_DEF is_a() {return SPLINEOBJECT;}
  SplineObject& operator=(SplineObject&);
  void set_object_ptrs(OperatorObject *_from_ptr,
                       OperatorObject *_to_ptr);
  void draw(StreamObject* parent, GC graphics_context,
            GC handle_context, DRAW_STYLE style,
            EXTERN_STATUS status);
  XYPAIR first_point();
  XYPAIR last_point();
  void draw_handles(GC draw_context, StreamObject *parent,
                    int x1, int y1);
  void clear();
  void add(int x, int y);
  void reset_iter();
  XYPAIR next_pair();
  BOOLEAN hit(int x, int y);
  void set_text_location(int &name_x, int &name_y,
                         int &latency_x, int &latency_y);
  void set_name_location(int &name_x, int &name_y);
  void set_latency_location(int name_x, int name_y,
                            int &latency_x, int &latency_y);
  void set_extern_location(XYPAIR &extern_location,
                           EXTERN_STATUS status);
  BOOLEAN hit_handle(int x, int y, EXTERN_STATUS status);
  void move_handle(int x, int y);
  BOOLEAN empty() {return _head_ptr == NULL;}
  void erase_handle(GC graphics_context, StreamObject *parent);
  void reset_handles_drawn() {_handles_drawn = FALSE;}
};

#endif
```

```
/* ************************************************************

Name:           spline_object.C
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 17 Sep 92
Remarks:    Implementation of the SplineObject class.

            The SplineObject is used to create the curved lines
          used in the graph_editor's splines.  It stores a
          linked list of control points which it uses to draw
          itself, point by point.

            In order to correctly terminate in the appropriate
          locations, it adds special control points I call
          'shadow points' at the ends of the lines.  Since
          b-splines don't normally end at the first and last
          control points, the extra shadow points make the
          curve stop in the right place.


Credits:     Portions of code are adapted from the following:

           Barakati, Naba, X Window System Programming, SAMS,
             1991.

           Heller, Dan, Motif Programming Manual, O'Reilly and
             Associates, 1991.

           Johnson, Eric, and Reichard, Kevin, X Window
             Applications Programming, MIS Press, 1989.

           Young, Douglas, Object Oriented Programming With C++
             and OSF/Motif, Prentice-Hall, 1992.

             Formula for b-spline curves comes from:

           Zyda, Michael, Book Number 5, Graphics and Video
             Laboratory, Naval Postgraduate School, 1990.

************************************************************ */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stream.h>
#include "ge_defs.h"
#include "spline_object.h"
#include "stream_object.h"
#include "operator_object.h"

#define ARROWANGLE 45.0
#define ARROWSIDELENGTH 10.0
#define HITDISTANCE 10
```

```
#define SKIPFACTOR 4
#define NONE 0
#define EXTERN_OFFSET 30

//   Initializes the static error target widget.

Widget SplineObject::_error_tgt = NULL;

//   Determines whether the string represents a valid number.

BOOLEAN SplineObject::valid_num_string(char *num) {
  int index, num_length;

  if(num != NULL) {
    num_length = strlen(num);
    if(num_length > 0) {
      for(index = 0; index < num_length; index++) {
        if((num[index] != '\n') &&
         ((num[index] < '0') || (num[index] > '9')))
          return FALSE;
      }
      return TRUE;
    }
  }
  return FALSE;
}

//   Displays the error message in a Motif error dialog box.

void SplineObject::error_box(char *error_message) {
  static Widget error_dialog = NULL;
  Arg arg[1];
  XmString t;

  if(_error_tgt != NULL) {
    if(error_dialog == NULL) {
      error_dialog = XmCreateMessageDialog(_error_tgt, "error",
                                           arg, 0);
      XtVaSetValues(XtParent(error_dialog),
                    XtNtitle, "Error",
                    NULL);
      XtUnmanageChild(XmMessageBoxGetChild(error_dialog,
                                           XmDIALOG_HELP_BUTTON));
    }
    t = XmStringCreateSimple(error_message);
    XtVaSetValues(error_dialog,
                  XmNmessageString, t,
                  NULL);
    XmStringFree(t);
    XtManageChild(error_dialog);
  }
  else
    cout <<
      "Error Target Widget must be set by calling module.\n"
```

```
        << endl;
}

SplineObject::SplineObject() {

  _head_ptr = NULL;
  _iter = NULL;
  _shadow_pts_set = FALSE;
  _handle_selected = NONE;
  _handles_drawn = FALSE;
}

//   Draws a handle at the given coordinates.

void SplineObject::draw_handles(GC draw_context,
                                StreamObject *parent, int x1,
                                int y1) {
#ifdef GEDEBUG
        cout << "Spline: " << (x1 - (HANDLESIZE / 2)) << " " <<
          (y1 - (HANDLESIZE / 2)) << " " << HANDLESIZE << endl;
#endif
        XSetFunction(parent->display_ptr(), draw_context, GXxor);
        XFillRectangle(parent->display_ptr(),
                       parent->draw_window(),
                       draw_context, (x1 - (HANDLESIZE / 2)),
                       (y1 - (HANDLESIZE / 2)), HANDLESIZE,
                       HANDLESIZE);
        XFillRectangle(parent->display_ptr(),
                       *(parent->drawing_area_pixmap()),
                       draw_context, (x1 - (HANDLESIZE / 2)),
                       (y1 - (HANDLESIZE / 2)), HANDLESIZE,
                       HANDLESIZE);
        XSetFunction(parent->display_ptr(), draw_context, GXcopy);
}

//   Builds the spline from disk.

GE_STATUS SplineObject::build_from_disk(FILE *infile) {
  char buffer[INPUT_LINE_SIZE + 1], *error_str_ptr = NULL;
  _spline_node *temp_node_ptr;
  int x, y;
  BOOLEAN done = FALSE;
  GE_STATUS status = SUCCEEDED;

  fgets(buffer, INPUT_LINE_SIZE, infile); // clear "SPLINE"
  while((!done) && (status == SUCCEEDED)) {
    if(strcmp("SPLINEEND\n",
              fgets(buffer, INPUT_LINE_SIZE, infile)) != 0) {
      if(valid_num_string(buffer))
        x = atoi(buffer);
      else
        error_str_ptr = strdup("Corrupted spline x");

      if(error_str_ptr == NULL) {
```

```
        fgets(buffer, INPUT_LINE_SIZE, infile);
        if(valid_num_string(buffer))
          y = atoi(buffer);
        else
          error_str_ptr = strdup("Corrupted spline y");
      }

      if(error_str_ptr == NULL) {
        if(_head_ptr == NULL) {
          _head_ptr = new _spline_node(x, y);
          temp_node_ptr = _head_ptr;
        }
        else {
          temp_node_ptr->_next_ptr = new _spline_node(x, y);
          temp_node_ptr = temp_node_ptr->_next_ptr;
        }
      }
    }
    else
      done = TRUE;
    if(error_str_ptr != NULL) {
      error_box(error_str_ptr);
      status = FAILED;
      free(error_str_ptr);
    }
  }
  if(ferror(infile)) {
    error_box("Unix reports an error building spline.");
    clearerr(infile);
    status = FAILED;
  }
  return status;
}

//   Writes the graphic attributes of the spline to disk.

GE_STATUS SplineObject::write_to_disk(FILE *outfile,
                                      EXTERN_STATUS status) {
  char buffer[INPUT_LINE_SIZE + 1];
  _spline_node *temp_node_ptr = _head_ptr;

  if(status == FROM_EXTERNAL)
    temp_node_ptr = temp_node_ptr->_next_ptr;
  else
    temp_node_ptr = temp_node_ptr->_next_ptr->_next_ptr;
  fprintf(outfile, "SPLINE\n");
  while(temp_node_ptr->_next_ptr->_next_ptr != NULL) {
    sprintf(buffer, "%d\n", temp_node_ptr->_x);
    fprintf(outfile, buffer);
    sprintf(buffer, "%d\n", temp_node_ptr->_y);
    fprintf(outfile, buffer);
    temp_node_ptr = temp_node_ptr->_next_ptr;
  }
  if(status == TO_EXTERNAL) {
```

```
      sprintf(buffer, "%d\n", temp_node_ptr->_x);
      fprintf(outfile, buffer);
      sprintf(buffer, "%d\n", temp_node_ptr->_y);
      fprintf(outfile, buffer);
    }
    fprintf(outfile, "SPLINEEND\n");
    if(ferror(outfile)) {
      clearerr(outfile);
      return FAILED;
    }
    else
      return SUCCEEDED;
  }

  //   Assignment operator.

  SplineObject& SplineObject::operator=(SplineObject& spline) {
    _spline_node *source_temp_ptr, *target_temp_ptr;

    if(spline._head_ptr != NULL) {
      _head_ptr = new _spline_node;
      _head_ptr->_x = spline._head_ptr->_x;
      _head_ptr->_y = spline._head_ptr->_y;
      source_temp_ptr = spline._head_ptr->_next_ptr;
      target_temp_ptr = _head_ptr;
      while(source_temp_ptr != NULL) {
        target_temp_ptr->_next_ptr = new _spline_node;
        target_temp_ptr = target_temp_ptr->_next_ptr;
        target_temp_ptr->_x = source_temp_ptr->_x;
        target_temp_ptr->_y = source_temp_ptr->_y;
        source_temp_ptr = source_temp_ptr->_next_ptr;
      }
    }
    return *this;
  }

  //   Returns the coordinates of the first point in the spline.

  XYPAIR SplineObject::first_point() {
    XYPAIR temp_pair;

    if(_head_ptr != NULL) {
      temp_pair.x = _head_ptr->_x;
      temp_pair.y = _head_ptr->_y;
    }
    else {
      temp_pair.x = 0;
      temp_pair.y = 0;
    }
    return temp_pair;
  }

  //   Returns the coordinates of the last point in the spline.
```

```
XYPAIR SplineObject::last_point() {
  XYPAIR temp_pair;
  _spline_node *temp_node_ptr = _head_ptr;

  if(_head_ptr != NULL) {
    while(temp_node_ptr->_next_ptr != NULL)
      temp_node_ptr = temp_node_ptr->_next_ptr;
    temp_pair.x = temp_node_ptr->_x;
    temp_pair.y = temp_node_ptr->_y;
  }
  else {
    temp_pair.x = 0;
    temp_pair.y = 0;
  }
  return temp_pair;
}

//   Sets _from_ptr and _to_ptr to the operators at either
// end of the spline.  Pointers for external streams equal
// NULL.  Also adds the necessary shadow and intercept points,
// and adds a control point for streams defined without one.
// Splines with externals must always have at least one point.

//   Streams with external endpoints use the last control
// point as the physical endpoint of the spline. Otherwise,
// the attached operator provides the coordinates for the
// intercept point, i.e. the spline endpoint.

//   In a spline with shadow points, the points are stored
// in the linked list in the following order:
//
//        'to' shadow point - 'to' intercept point -
//                 (defined control points) -
//        'from' intercept point - 'from' shadow point

void SplineObject::set_object_ptrs(OperatorObject *_from_ptr,
                                   OperatorObject *_to_ptr) {
  XYPAIR from_intercept, to_intercept, temp_pair, first_point,
    last_point;
  _spline_node *temp_node_ptr, *temp_head_ptr, *temp_node_ptr2;

  if(_shadow_pts_set == TRUE) {

//   strips off existing intercept and shadow points. For
// externals, the intercept points are saved.

    temp_node_ptr = _head_ptr;
    while(temp_node_ptr->_next_ptr->_next_ptr->_next_ptr != NULL)
      temp_node_ptr = temp_node_ptr->_next_ptr;
    if(_to_ptr == NULL) {
      temp_pair.x = temp_node_ptr->_next_ptr->_x;
      temp_pair.y = temp_node_ptr->_next_ptr->_y;
      to_intercept = temp_pair;
    }
```

```
  delete temp_node_ptr->_next_ptr;
  temp_node_ptr->_next_ptr = NULL;
  temp_node_ptr2 = _head_ptr;
  temp_node_ptr = _head_ptr->_next_ptr->_next_ptr;
  temp_node_ptr2->_next_ptr->_next_ptr = NULL;
  if(_from_ptr == NULL) {
    temp_pair.x = _head_ptr->_next_ptr->_x;
    temp_pair.y = _head_ptr->_next_ptr->_y;
    from_intercept = temp_pair;
  }
  delete temp_node_ptr2;
  _head_ptr = temp_node_ptr;
}
else {
  _shadow_pts_set = TRUE;
  if(_from_ptr == NULL) {
    if(_head_ptr == NULL) {
      error_box("External streams must have at least one control point");
      return;
    }
    else {
      temp_node_ptr = _head_ptr;
      _head_ptr = _head_ptr->_next_ptr;
      from_intercept.x = temp_node_ptr->_x;
      from_intercept.y = temp_node_ptr->_y;
      temp_node_ptr->_next_ptr = NULL;
      delete temp_node_ptr;
    }
  }
  if(_to_ptr == NULL) {
    if(_head_ptr == NULL)
      error_box("External streams must have at least one control point");
    else {
      temp_node_ptr = _head_ptr;
      if(temp_node_ptr->_next_ptr == NULL) {
        to_intercept.x = temp_node_ptr->_x;
        to_intercept.y = temp_node_ptr->_y;
        delete temp_node_ptr;
        _head_ptr = NULL;
      }
      else {
        while(temp_node_ptr->_next_ptr->_next_ptr != NULL)
          temp_node_ptr = temp_node_ptr->_next_ptr;
        to_intercept.x = temp_node_ptr->_next_ptr->_x;
        to_intercept.y = temp_node_ptr->_next_ptr->_y;
        delete temp_node_ptr->_next_ptr;
        temp_node_ptr->_next_ptr = NULL;
      }
    }
  }
}
if(_head_ptr != NULL) {
  temp_pair = this->first_point();
  if(_from_ptr != NULL)
```

```
    from_intercept = _from_ptr->intercept(temp_pair.x,
                                          temp_pair.y);
  first_point.x = from_intercept.x -
    (temp_pair.x - from_intercept.x);
  first_point.y = from_intercept.y -
    (temp_pair.y - from_intercept.y);
  temp_pair = this->last_point();
  if(_to_ptr != NULL)
    to_intercept = _to_ptr->intercept(temp_pair.x,
                                      temp_pair.y);
  last_point.x = to_intercept.x - (temp_pair.x -
                                   to_intercept.x);
  last_point.y = to_intercept.y - (temp_pair.y -
                                   to_intercept.y);
  temp_head_ptr = new _spline_node(first_point);
  temp_head_ptr->_next_ptr = new _spline_node(from_intercept);
  temp_node_ptr = temp_head_ptr->_next_ptr;
  temp_node_ptr->_next_ptr = _head_ptr;
  while(temp_node_ptr->_next_ptr != NULL)
    temp_node_ptr = temp_node_ptr->_next_ptr;
  temp_node_ptr->_next_ptr = new _spline_node(to_intercept);
  temp_node_ptr->_next_ptr->_next_ptr =
    new _spline_node(last_point);
  _head_ptr = temp_head_ptr;
}
else {
  if(_to_ptr == NULL)
    from_intercept = _from_ptr->intercept(to_intercept.x,
                                          to_intercept.y);
  else
    if(_from_ptr == NULL)
      to_intercept =
        _to_ptr->intercept(from_intercept.x,
                           from_intercept.y);
    else {
      temp_pair = _to_ptr->center();
      from_intercept = _from_ptr->intercept(temp_pair.x,
                                            temp_pair.y);
      to_intercept = _to_ptr->intercept(from_intercept.x,
                                        from_intercept.y);
    }
  temp_pair.x = (from_intercept.x + to_intercept.x) / 2;
  temp_pair.y = (from_intercept.y + to_intercept.y) / 2;
  first_point.x = from_intercept.x - (temp_pair.x -
                                      from_intercept.x);
  first_point.y = from_intercept.y - (temp_pair.y -
                                      from_intercept.y);
  last_point.x = to_intercept.x - (temp_pair.x -
                                   to_intercept.x);
  last_point.y = to_intercept.y - (temp_pair.y -
                                   to_intercept.y);
  temp_head_ptr = new _spline_node(first_point);
  temp_head_ptr->_next_ptr = new _spline_node(from_intercept);
  temp_node_ptr = temp_head_ptr->_next_ptr;
```

```
    temp_node_ptr->_next_ptr = new _spline_node(temp_pair);
    temp_node_ptr = temp_node_ptr->_next_ptr;
    temp_node_ptr->_next_ptr = new _spline_node(to_intercept);
    temp_node_ptr->_next_ptr->_next_ptr =
      new _spline_node(last_point);
    _head_ptr = temp_head_ptr;
  }
}

//   Determines the number of points to be calculated for the
// spline.

int SplineObject::num_points(_spline_node *p1, _spline_node *p2,
                             _spline_node *p3,
                             _spline_node *p4) {
  int distance, number_points = 0;

  if(p4 != NULL) {
    distance = abs(p1->_x - p2->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_x - p3->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_x - p4->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p2->_x - p3->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p2->_x - p4->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p3->_x - p4->_x);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_y - p2->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_y - p3->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p1->_y - p4->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p2->_y - p3->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p2->_y - p4->_y);
    if(distance > number_points)
      number_points = distance;
    distance = abs(p3->_y - p4->_y);
    if(distance > number_points)
      number_points = distance;
```

```
  }
  return number_points;
}

//   Draws the arrowheads for the line.

void SplineObject::draw_arrowhead(GC graphics_context,
                                  StreamObject *parent,
                                  _spline_node *endpoint) {
  _spline_node *last_point = endpoint->_next_ptr;
  double angle, temp_angle;
  double half_arrow_angle = ARROWANGLE / 2.0 * M_PI / 180.0;
  XPoint vertices[3];

  if(last_point->_x == endpoint->_x) {
    if(last_point->_y > endpoint->_y)
      angle = M_PI / 2.0;
    else
      angle = 3.0 * M_PI / 2.0;
  }
  else {
    angle = atan((double) (last_point->_y - endpoint->_y) /
                 (double) (last_point->_x - endpoint->_x));
    if((last_point->_x < endpoint->_x))
      angle = M_PI + angle;
  }
  vertices[0].x = endpoint->_x;
  vertices[0].y = endpoint->_y;
  temp_angle = angle - half_arrow_angle;
  vertices[1].x = (short) (endpoint->_x - (cos(temp_angle)
                                           * ARROWSIDELENGTH));
  vertices[1].y = (short) (endpoint->_y - (sin(temp_angle) *
                                           ARROWSIDELENGTH));
  temp_angle = angle + half_arrow_angle;
  vertices[2].x = (short) (endpoint->_x - (cos(temp_angle) *
                                           ARROWSIDELENGTH));
  vertices[2].y = (short) (endpoint->_y - (sin(temp_angle) *
                                           ARROWSIDELENGTH));
  XFillPolygon(parent->display_ptr(), parent->draw_window(),
               graphics_context, vertices, 3, Convex,
               CoordModeOrigin);
  XFillPolygon(parent->display_ptr(),
               *(parent->drawing_area_pixmap()),
               graphics_context, vertices, 3, Convex,
               CoordModeOrigin);
}

//   Draws the spline point by point, and if the coordinates
// are near a spline point, TRUE is returned.

BOOLEAN SplineObject::hit(int in_x, int in_y) {
  int points;
  float inc;
  float t, t2, t3, term1, term2, term3, term4, x, y;
```

```
  _spline_node *p1, *p2, *p3, *p4;

  p1 = _head_ptr;
  p2 = p1->_next_ptr;
  p3 = p2->_next_ptr;
  p4 = p3->_next_ptr;
  points = num_points(p1, p2, p3, p4) / SKIPFACTOR;
  while(p4 != NULL) {
    inc = 1.0 / (float) points;
    for(t = 0.0; t <= 1.0; t += inc) {
      t2 = t * t;
      t3 = t2 * t;
      term1 = (-t3 + (3 * t2) - (3 * t) +1);
      term2 = ((3 * t3) - (t2 * 6) + 4);
      term3 = ((-3 * t3) + (3 * t2) + (3 * t) + 1);
      term4 = t3;
      x =  term1 * p1->_x;
      x += term2 * p2->_x;
      x += term3 * p3->_x;
      x += term4 * p4->_x;
      x /= 6.0;

      y =  term1 * p1->_y;
      y += term2 * p2->_y;
      y += term3 * p3->_y;
      y += term4 * p4->_y;
      y /= 6.0;
      if((abs((int) x - in_x) < HITDISTANCE) &&
         (abs((int) y - in_y) < HITDISTANCE))
        return TRUE;
    }
    p1 = p2;
    p2 = p3;
    p3 = p4;
    p4 = p4->_next_ptr;
    points = num_points(p1, p2, p3, p4) / SKIPFACTOR;
  }
  return FALSE;
}

//   Draws the spline using the control points.

void SplineObject::draw(StreamObject *parent,
                        GC graphics_context, GC handle_context,
                        DRAW_STYLE style,
                        EXTERN_STATUS status) {
  int points, i, j;
  float inc, t, t2, t3, term1, term2, term3, term4, x, y;
  _spline_node *p1, *p2, *p3, *p4, *temp_node_ptr;
  BOOLEAN need_handles;

  p1 = _head_ptr;
  p2 = p1->_next_ptr;
  p3 = p2->_next_ptr;
```

```
  p4 = p3->_next_ptr;
  points = num_points(p1, p2, p3, p4);
  while(p4 != NULL) {
/* Debug  code
    if((p1->_x > 1000) || (p1->_x < 0))
      printf("Bogus draw p1x: %d id: %d", p1->_x, parent->id());
    if((p1->_y > 1000) || (p1->_y < 0))
      printf("Bogus draw p1y: %d id: %d", p1->_y, parent->id());
    if((p2->_x > 1000) || (p2->_x < 0))
      printf("Bogus draw p2x: %d id: %d", p2->_x, parent->id());
    if((p2->_y > 1000) || (p2->_y < 0))
      printf("Bogus draw p2y: %d id: %d", p2->_y, parent->id());
    if((p3->_x > 1000) || (p3->_x < 0))
      printf("Bogus draw p3x: %d id: %d", p3->_x, parent->id());
    if((p3->_y > 1000) || (p3->_y < 0))
      printf("Bogus draw p3y: %d id: %d", p3->_y, parent->id());
    if((p4->_x > 1000) || (p4->_x < 0))
      printf("Bogus draw p4x: %d id: %d", p4->_x, parent->id());
    if((p4->_y > 1000) || (p4->_y < 0))
      printf("Bogus draw p4y: %d id: %d", p4->_y, parent->id());
*/
    inc = 1.0 / (float) points;
    for(t = 0.0; t <= 1.0; t += inc) {
      t2 = t * t;
      t3 = t2 * t;
      term1 = (-t3 + (3 * t2) - (3 * t) +1);
      term2 = ((3 * t3) - (t2 * 6) + 4);
      term3 = ((-3 * t3) + (3 * t2) + (3 * t) + 1);
      term4 = t3;
      x =  term1 * p1->_x;
      x += term2 * p2->_x;
      x += term3 * p3->_x;
      x += term4 * p4->_x;
      x /= 6.0;

      y =  term1 * p1->_y;
      y += term2 * p2->_y;
      y += term3 * p3->_y;
      y += term4 * p4->_y;
      y /= 6.0;
      if(parent->is_state_variable()) { // draw thicker line
        for(i = (int) x - 1; i < (int) x + 1; i++)
          for(j = (int) y - 1; j < (int) y + 1; j++) {
            XDrawPoint(parent->display_ptr(),
                       parent->draw_window(),
                       graphics_context, i, j);
            XDrawPoint(parent->display_ptr(),
                       *(parent->drawing_area_pixmap()),
                       graphics_context, i, j);
          }
      }
      else {
        XDrawPoint(parent->display_ptr(), parent->draw_window(),
                   graphics_context, (int) x, (int) y);
```

```
        XDrawPoint(parent->display_ptr(),
                   *(parent->drawing_area_pixmap()),
                   graphics_context, (int) x, (int) y);
      }
    }
    p1 = p2;
    p2 = p3;
    p3 = p4;
    p4 = p4->_next_ptr;
    points = num_points(p1, p2, p3, p4);
  }
  temp_node_ptr = _head_ptr->_next_ptr;
  if(((_handles_drawn == FALSE) && (style == SOLID)) ||
      ((_handles_drawn == TRUE) && (style == ERASE)))
    need_handles = TRUE;
  else
    need_handles = FALSE;

  if(parent->is_selected() && (status == FROM_EXTERNAL) &&
     need_handles)
      draw_handles(handle_context, parent, temp_node_ptr->_x,
                   temp_node_ptr->_y);
  temp_node_ptr = temp_node_ptr->_next_ptr;
  while(temp_node_ptr->_next_ptr->_next_ptr != NULL) {
    if(parent->is_selected() && need_handles)
      draw_handles(handle_context, parent, temp_node_ptr->_x,
                   temp_node_ptr->_y);
    temp_node_ptr = temp_node_ptr->_next_ptr;
  }
  draw_arrowhead(graphics_context, parent, temp_node_ptr);
  if(parent->is_selected() && (status == TO_EXTERNAL) &&
     need_handles)
    draw_handles(handle_context, parent, temp_node_ptr->_x,
                 temp_node_ptr->_y);
  if(need_handles) {
    if(_handles_drawn == TRUE)
      _handles_drawn = FALSE;
    else
      _handles_drawn = TRUE;
  }
}

//   Erases the control points.

void SplineObject::clear() {

  delete _head_ptr;
  _head_ptr = NULL;
}

//   Adds a new control point to the spline.

void SplineObject::add(int x, int y) {
  _spline_node *temp_node_ptr;
```

```
/* Debug code
  if((x > 900) || (x < 0))
    printf("Bogus spline x: %d", x);
  if((y > 900) || (y < 0))
    printf("Bogus spline y: %d", x);
*/
  if(_head_ptr == NULL)
    _head_ptr = new _spline_node(x, y);
  else {
    temp_node_ptr = _head_ptr;
    while(temp_node_ptr->_next_ptr != NULL)
      temp_node_ptr = temp_node_ptr->_next_ptr;
    temp_node_ptr->_next_ptr = new _spline_node(x, y);
  }
}

//   Returns the control point pointed to by the iterator
// pointer, then advances the iterator pointer.

XYPAIR SplineObject::next_pair() {
  XYPAIR temp_pair;

  if(_iter == NULL) {
    temp_pair.x = -1;
    temp_pair.y = -1;
  }
  else {
    temp_pair.x = _iter->_x;
    temp_pair.y = _iter->_y;
    _iter = _iter->_next_ptr;
  }
  return temp_pair;
}

//   Resets the iterator pointer to the beginning of the spline.

void SplineObject::reset_iter() {

  _iter = _head_ptr;
}

//   Places the name between the middle two control points.

void SplineObject::set_name_location(int &name_x, int &name_y) {

  _spline_node *temp_node_ptr = _head_ptr;
  int temp_x, temp_y, i, num_nodes = 0;

  while(temp_node_ptr != NULL) {
    num_nodes++;
    temp_node_ptr = temp_node_ptr->_next_ptr;
  }
  temp_node_ptr = _head_ptr;
```

```
  if(_head_ptr != NULL) {
    for(i = 1; i < num_nodes / 2; i++)
      temp_node_ptr = temp_node_ptr->_next_ptr;
    temp_x =
      (temp_node_ptr->_x + temp_node_ptr->_next_ptr->_x) / 2;
    temp_y =
      (temp_node_ptr->_y + temp_node_ptr->_next_ptr->_y) / 2;
    name_x = temp_x;
    name_y = temp_y;
  }
}

//   Places the latency location underneath the name.

void SplineObject::set_latency_location(int temp_x, int temp_y,
                                        int &latency_x,
                                        int &latency_y) {

  latency_x = temp_x;
  latency_y = temp_y + 15;
}

//   Convenience function for setting the text location.

void SplineObject::set_text_location(int &name_x, int &name_y,
                                     int &latency_x,
                                     int &latency_y) {

  set_name_location(name_x, name_y);
  set_latency_location(name_x, name_y, latency_x, latency_y);
}

//   Checks to see if the coordinates are within any of the
// handles, or within the ends of external streams.

BOOLEAN SplineObject::hit_handle(int x, int y,
                                 EXTERN_STATUS status) {
  int num_handle = 2;
  _spline_node *temp_node_ptr = _head_ptr->_next_ptr;

  if(status != FROM_EXTERNAL) {
    temp_node_ptr = temp_node_ptr->_next_ptr;
    num_handle++;
  }
  while(temp_node_ptr->_next_ptr->_next_ptr != NULL) {
    if((((temp_node_ptr->_x) - (HANDLESIZE / 2) - HITFUDGE)
                                                    <= x) &&
       (x <=
   ((temp_node_ptr->_x) + (HANDLESIZE / 2) + HITFUDGE)) &&
   (((temp_node_ptr->_y) - (HANDLESIZE / 2) - HITFUDGE) <= y) &&
   (y <= ((temp_node_ptr->_y) + (HANDLESIZE / 2) + HITFUDGE))) {
         _handle_selected = num_handle;
         return TRUE;
    }
```

```
    temp_node_ptr = temp_node_ptr->_next_ptr;
    num_handle++;
  }
  if(status == TO_EXTERNAL) {
    if((((temp_node_ptr->_x) - (HANDLESIZE / 2) - HITFUDGE)
                                                    <= x) &&
       (x <=
          ((temp_node_ptr->_x) + (HANDLESIZE / 2) + HITFUDGE)) &&
           (((temp_node_ptr->_y) - (HANDLESIZE / 2) - HITFUDGE)
            <= y) &&
          (y <= ((temp_node_ptr->_y) + (HANDLESIZE / 2) +
                HITFUDGE))) {
        _handle_selected = num_handle;
        return TRUE;
    }
  }
  _handle_selected = NONE;
  return FALSE;
}

//   Moves a spline handle.

void SplineObject::move_handle(int x, int y) {
  _spline_node *temp_node_ptr = _head_ptr;
  int i;

  for(i = 1; i < _handle_selected; i++)
    temp_node_ptr = temp_node_ptr->_next_ptr;
  temp_node_ptr->_x += x;
  temp_node_ptr->_y += y;
  if(temp_node_ptr->_x < 0)
    temp_node_ptr->_x = 0;
  if(temp_node_ptr->_y < 0)
    temp_node_ptr->_y = 0;
}

//   Erases the given handle

void SplineObject::erase_handle(GC graphics_context,
                                StreamObject *parent) {
  _spline_node *temp_node_ptr = _head_ptr;
  int i;

  for(i = 1; i < _handle_selected; i++)
    temp_node_ptr = temp_node_ptr->_next_ptr;
  draw_handles(graphics_context, parent, temp_node_ptr->_x,
               temp_node_ptr->_y);
}

//   Determines the location for the 'EXTERNAL' label on
// the drawing.

void SplineObject::set_extern_location(XYPAIR &extern_location,
                       EXTERN_STATUS status) {
```

```
  _spline_node *temp_node_ptr, *intercept_ptr, *shadow_ptr;
  int temp_x, temp_y;

  if(status == FROM_EXTERNAL) {
    shadow_ptr = _head_ptr;
    intercept_ptr = _head_ptr->_next_ptr;
  }
  else
    if(status == TO_EXTERNAL) {
      temp_node_ptr = _head_ptr;
      while(temp_node_ptr->_next_ptr->_next_ptr != NULL)
        temp_node_ptr = temp_node_ptr->_next_ptr;
      intercept_ptr = temp_node_ptr;
      shadow_ptr = temp_node_ptr->_next_ptr;
    }
  if(shadow_ptr->_y > intercept_ptr->_y)
    temp_y = intercept_ptr->_y + 7;
  else
    temp_y = intercept_ptr->_y - 7;
  if(shadow_ptr->_y == intercept_ptr->_y)
    if(shadow_ptr->_x > intercept_ptr->_x)
      temp_x = EXTERN_OFFSET;
    else
      temp_x = -EXTERN_OFFSET;
  else
    temp_x = (temp_y - intercept_ptr->_y) /
      ((shadow_ptr->_y) - (intercept_ptr->_y)) *
        ((shadow_ptr->_x) - (intercept_ptr->_x));
  if(abs(temp_x) > EXTERN_OFFSET)
    if(shadow_ptr->_x > intercept_ptr->_x)
      temp_x = EXTERN_OFFSET;
    else
      temp_x = -EXTERN_OFFSET;
  temp_x += intercept_ptr->_x;
  temp_y += 10;
  extern_location.x = temp_x;
  extern_location.y = temp_y;
}
```

```
/* ************************************************************

Name:           stream.h
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 11 Sep 92
Remarks:   Specification for the StreamObject class.

           The StreamObject is a graphical representation of
         a curved PSDL stream.  The stream is drawn as a
         b-spline specified by a series of control points.
         Streams automatically connect to their attached
         operators, and always have arrowheads at their
         terminating points.

************************************************************* */
#ifndef stream_object_h
#define stream_object_h 1

#include <stdio.h>
#include <X11/Xlib.h>
#include "ge_defs.h"
#include "graph_object.h"
#include "spline_object.h"
#include "operator_object.h"

class GraphObjectList;

class StreamObject : public GraphObject {
protected:
  char *_name_ptr, *_latency_string_ptr;
  int _id, _from, _to, _latency, _name_font, _name_x, _name_y,
  _latency_font, _latency_x, _latency_y, _name_height,
  _name_width, _latency_width, _latency_height,
  _handle_selected;
  SplineObject _arc;
  BOOLEAN _is_deleted, _is_new, _is_modified, _is_state_variable,
    _is_selected, _name_selected, _latency_selected,
    _st_handles_drawn, _name_handles_drawn,
    _latency_handles_drawn;
  OperatorObject *_from_ptr, *_to_ptr;
  XYPAIR name_location, lat_location, extern_location;

  void set_default_text_location();
  void set_default_name_location();
  void set_default_latency_location();
  void StreamObject::draw_handles(GC draw_context, int x1,
                                  int y1, int x2, int y2);
public:
  StreamObject();
  StreamObject(char *in_name_ptr, int in_id, int in_from,
               int in_to, int in_latency, SplineObject in_arc,
               BOOLEAN in_is_new, BOOLEAN in_is_state_variable);
  ~StreamObject() {delete _name_ptr;delete _latency_string_ptr;}
```

```
  GE_STATUS build_from_property();
  GE_STATUS build_from_disk(FILE *infile);
  GE_STATUS write_to_property();
  GE_STATUS write_to_disk(FILE *outfile);
  void draw_spline(DRAW_STYLE style);
  void draw(DRAW_STYLE style);
  void draw_text(DRAW_STYLE style);
  void erase_text();
  void move_text(int x, int y);
  void select();
  void unselect();
  void erase();
  BOOLEAN hit(int x, int y);
  CLASS_DEF is_a() {return STREAMOBJECT;}
  void set_object_ptrs(GraphObjectList *parent);
  int id() {return _id;}
  char *name() {return _name_ptr;}
  void set_deleted() {_is_deleted = TRUE;}
  void delete_notify(CLASS_DEF class_type, int deleted_obj_id);
  void replace_name(char *new_name);
  void set_constraint(int constraint);
  void move_notify(CLASS_DEF object_type, int object_id);
  BOOLEAN is_selected() {return _is_selected;}
  BOOLEAN is_state_variable() {return _is_state_variable;}
  void set_state_variable(BOOLEAN state)
                                    {_is_state_variable = state;}
  BOOLEAN hit_handle(int x, int y);
  void move_handle(int x, int y) {_arc.move_handle(x, y);}
  void set_modified() {_is_modified = TRUE;}
  BOOLEAN text_selected()
                {return (_name_selected || _latency_selected);}
  void set_text_dimensions();
  void set_object_font(int font_id);
  void undelete_notify(CLASS_DEF class_type, int deleted_obj_id);
  void reset_handles_drawn_state();
  int text_width();
  int text_height();
  void text_locate(int x, int y);
  void erase_handle() {_arc.erase_handle(_graphics_context, this);}
  int constraint() {return _latency;}
};

#endif;
```

```
/* ************************************************************

Name:           stream_object.C
Author:         Capt Robert M. Dixon
Program:        graph_editor
Date Modified: 11 Sep 92
Remarks:    Implementation of the StreamObject class.

            The StreamObject is a graphical representation of
          a curved PSDL stream.  The stream is drawn as a
          b-spline specified by a series of control points.
          Streams automatically connect to their attached
          operators, and always have arrowheads at their
          terminating points.


Credits:     Portions of code are adapted from the following:

           Barakati, Naba, X Window System Programming, SAMS,
             1991.

           Heller, Dan, Motif Programming Manual, O'Reilly and
             Associates, 1991.

           Johnson, Eric, and Reichard, Kevin, X Window
             Applications Programming, MIS Press, 1989.

           Young, Douglas, Object Oriented Programming With C++
             and OSF/Motif, Prentice-Hall, 1992.

************************************************************* */
#include <stdlib.h>
#include <string.h>
#include <stream.h>
#include "stream_object.h"
#include "graph_object_list.h"

#define MAX_CONTROL_POINTS 100

StreamObject::StreamObject() : GraphObject() {

  _name_ptr = NULL;
  _latency_string_ptr = NULL;
  _from_ptr = NULL;
  _to_ptr = NULL;
  _is_selected = FALSE;
  _name_font = _default_font;
  _latency_font = _default_font;
  _name_x = NULL_VALUE;
  _name_y = NULL_VALUE;
  _latency_x = NULL_VALUE;
  _latency_y = NULL_VALUE;
  _name_selected = FALSE;
  _latency_selected = FALSE;
```

```
  set_text_dimensions();
  reset_handles_drawn_state();
}

//   Latency values less than zero are assumed to refer to
// microseconds.

StreamObject::StreamObject(char *in_name_ptr, int in_id,
    int in_from, int in_to, int in_latency,
    SplineObject in_arc, BOOLEAN in_is_new,
    BOOLEAN in_is_state_variable) : GraphObject() {
  char buffer[INPUT_LINE_SIZE];

  _name_ptr = strdup(in_name_ptr);
  _name_font = _default_font;
  _id = in_id;
  _from = in_from;
  _to = in_to;
  _latency = in_latency;
  if(_latency == NULL_VALUE)
    _latency_string_ptr = NULL;
  else {
    if(_latency < 0)
      sprintf(buffer, "%d us", -_latency);
    else
      sprintf(buffer, "%d ms", _latency);
    _latency_string_ptr = strdup(buffer);
  }

  _latency_font = _default_font;
  _arc = in_arc;
  _is_state_variable = in_is_state_variable;

  _is_deleted = FALSE;
  _is_new = in_is_new;
  _is_modified = FALSE;
  _is_selected = FALSE;
  _name_selected = FALSE;
  _latency_selected = FALSE;
  _from_ptr = NULL;
  _to_ptr = NULL;
  set_text_dimensions();
  reset_handles_drawn_state();
}

GE_STATUS StreamObject::build_from_property() {

  return FAILED;
}

//   Builds the stream from disk.

GE_STATUS StreamObject::build_from_disk(FILE *infile) {
  char buffer[INPUT_LINE_SIZE + 1], *error_str_ptr = NULL;
```

```
int last_char;
GE_STATUS status = SUCCEEDED;

fgets(buffer, INPUT_LINE_SIZE, infile);
last_char = strlen(buffer) - 1;
if(buffer[last_char] == '\n')
  buffer[last_char] = 0;
if(strcmp(buffer, "ENDDATA") == 0)
  return ENDED;
else {
  _name_ptr = strdup(buffer);

  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer)) {
    _name_font = atoi(buffer);
    if(_name_font > MAXFONTS)
      error_str_ptr = strdup("Name Font Too Large");
  }
  else
    error_str_ptr = strdup("Corrupted Name Font");

  if(error_str_ptr == NULL) {
    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer))
      _name_x = atoi(buffer);
    else
      error_str_ptr = strdup("Corrupted name_x");
  }

  if(error_str_ptr == NULL) {
    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer))
      _name_y = atoi(buffer);
    else
      error_str_ptr = strdup("Corrupted name_y");
  }

  if(error_str_ptr == NULL) {
    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer))
      _id = atoi(buffer);
    else
      error_str_ptr = strdup("Corrupted id");
  }

  if(error_str_ptr == NULL) {
    fgets(buffer, INPUT_LINE_SIZE, infile);
    if(valid_num_string(buffer))
      _from = atoi(buffer);
    else
      error_str_ptr = strdup("Corrupted \"from\"");
  }

  if(error_str_ptr == NULL) {
```

```
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer)) {
    _to = atoi(buffer);
    if((_from == 0) && (_to == 0))
      error_str_ptr =
        strdup("From & To pointers both NULL.\n");
  }
  else
    error_str_ptr = strdup("Corrupted \"to\"");
}

if(error_str_ptr == NULL) {
  status = _arc.build_from_disk(infile);
  if(status == FAILED)
    error_str_ptr = strdup("Unable to build spline");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _latency = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted latency");
}

if(error_str_ptr == NULL) {
  if(_latency == NULL_VALUE)
    _latency_string_ptr = NULL;
  else {
    if(_latency < 0)
      sprintf(buffer, "%d us", -_latency);
    else
      sprintf(buffer, "%d ms", _latency);
    _latency_string_ptr = strdup(buffer);
  }
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer)) {
    _latency_font = atoi(buffer);
    if(_latency_font > MAXFONTS)
      error_str_ptr = strdup("Latency Font Too Large");
  }
  else
    error_str_ptr = strdup("Corrupted Latency Font");
}

if(error_str_ptr == NULL) {
  fgets(buffer, INPUT_LINE_SIZE, infile);
  if(valid_num_string(buffer))
    _latency_x = atoi(buffer);
  else
    error_str_ptr = strdup("Corrupted latency_x");
```

```
    }

    if(error_str_ptr == NULL) {
      fgets(buffer, INPUT_LINE_SIZE, infile);
      if(valid_num_string(buffer))
        _latency_y = atoi(buffer);
      else
        error_str_ptr = strdup("Corrupted latency_y");
    }

    if(error_str_ptr == NULL) {
      if(strcmp("TRUE\n",
         fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
        _is_deleted = TRUE;
      else
        if(strcmp("FALSE\n", buffer) == 0)
          _is_deleted = FALSE;
        else
          error_str_ptr = strdup("Corrupted is_deleted");
    }

    if(error_str_ptr == NULL) {
      if(strcmp("TRUE\n",
         fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)

//Change this if original _is_new should be saved.

        _is_new = FALSE;
        else
          if(strcmp("FALSE\n", buffer) == 0)
            _is_new = FALSE;
          else
            error_str_ptr = strdup("Corrupted is_new");
    }

    if(error_str_ptr == NULL) {
      if(strcmp("TRUE\n",
         fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
        _is_modified = TRUE;
      else
        if(strcmp("FALSE\n", buffer) == 0)
          _is_modified = FALSE;
        else
          error_str_ptr = strdup("Corrupted is_modified");
    }

    if(error_str_ptr == NULL) {
      if(strcmp("TRUE\n",
         fgets(buffer, INPUT_LINE_SIZE, infile)) == 0)
        _is_state_variable = TRUE;
      else
        if(strcmp("FALSE\n", buffer) == 0)
          _is_state_variable = FALSE;
        else
```

```
            error_str_ptr = strdup("Corrupted is_state_variable");
        }

        _is_selected = FALSE;
        _name_selected = FALSE;
        _latency_selected = FALSE;

        if(error_str_ptr == NULL) {
          set_text_dimensions();
        }

        if(error_str_ptr != NULL) {
          status = FAILED;
          sprintf(buffer, "Stream %s: %s", _name_ptr, error_str_ptr);
          error_box(buffer);
          free(error_str_ptr);
        }

        if(ferror(infile)) {
          sprintf(buffer,
            "Unix reported an error building stream %s", _name_ptr);
          error_box(buffer);
          clearerr(infile);
          status = FAILED;
        }
      }
      return status;
}

//   Draws handles around the specified location.  Handles are
// drawn in exclusive-or mode to simplify erasing them without
// disturbing the underlying text.

void StreamObject::draw_handles(GC draw_context, int x1, int y1,
                                int x2, int y2) {

  XSetFunction(_display_ptr, draw_context, GXxor);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context, x1,
                 y2 - HANDLESIZE, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, _draw_window, draw_context,
                 x2 - HANDLESIZE, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y1, HANDLESIZE, HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y1, HANDLESIZE,
                 HANDLESIZE);
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x1, y2 - HANDLESIZE, HANDLESIZE,
                 HANDLESIZE);
```

```
  XFillRectangle(_display_ptr, *_drawing_area_pixmap,
                 draw_context, x2 - HANDLESIZE, y2 - HANDLESIZE,
                 HANDLESIZE, HANDLESIZE);
  XSetFunction(_display_ptr, draw_context, GXcopy);
}

//   Draws text strings.

void StreamObject::draw_text(DRAW_STYLE style) {
  GC draw_context;

  if(style == SOLID)
    draw_context = _graphics_context;
  else
    if(style == ERASE)
      draw_context = _erase_context;
  set_font(draw_context, _name_font);
  XDrawString(_display_ptr, _draw_window, draw_context,
              _name_x, _name_y, _name_ptr, strlen(_name_ptr));
  XDrawString(_display_ptr, *_drawing_area_pixmap, draw_context,
              _name_x, _name_y, _name_ptr, strlen(_name_ptr));
  if(_name_selected) {
    if((_name_handles_drawn == FALSE) && (style == SOLID)) {
      draw_handles(_graphics_context, _name_x  - HANDLESIZE,
                   _name_y - _name_height - HANDLESIZE,
                   _name_x + _name_width + HANDLESIZE,
                   _name_y + HANDLESIZE);
      _name_handles_drawn = TRUE;
    }
    else
      if((_name_handles_drawn == TRUE) && (style == ERASE)) {
        draw_handles(_graphics_context, _name_x  - HANDLESIZE,
                     _name_y - _name_height - HANDLESIZE,
                     _name_x + _name_width + HANDLESIZE,
                     _name_y + HANDLESIZE);
        _name_handles_drawn = FALSE;
      }
  }

  if(_latency != NULL_VALUE) {
    set_font(draw_context, _latency_font);
    XDrawString(_display_ptr, _draw_window, draw_context,
                _latency_x, _latency_y, _latency_string_ptr,
                strlen(_latency_string_ptr));
    XDrawString(_display_ptr, *_drawing_area_pixmap,
                draw_context, _latency_x, _latency_y,
                _latency_string_ptr,
                strlen(_latency_string_ptr));
    if(_latency_selected) {
      if((_latency_handles_drawn == FALSE) && (style == SOLID)) {
        draw_handles(_graphics_context, _latency_x  - HANDLESIZE,
                     _latency_y - _latency_height - HANDLESIZE,
                     _latency_x + _latency_width + HANDLESIZE,
                     _latency_y + HANDLESIZE);
```

```
        _latency_handles_drawn = TRUE;
      }
      else
        if((_latency_handles_drawn == TRUE) &&
           (style == ERASE)) {
          draw_handles(_graphics_context,
                       _latency_x  - HANDLESIZE,
                       _latency_y - _latency_height - HANDLESIZE,
                       _latency_x + _latency_width + HANDLESIZE,
                       _latency_y + HANDLESIZE);
          _latency_handles_drawn = FALSE;
        }
    }
  }
}

void StreamObject::erase_text() {

  draw_text(ERASE);
}

GE_STATUS StreamObject::write_to_property() {
  return FAILED;
}

//   Writes the graphic attributes of the stream to disk.

GE_STATUS StreamObject::write_to_disk(FILE *outfile) {
  char buffer[INPUT_LINE_SIZE + 1];

  fprintf(outfile, "%s\n", _name_ptr);
  sprintf(buffer, "%d\n%d\n%d", _name_font, _name_x, _name_y);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", _id);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", _from);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d", _to);
  fprintf(outfile, "%s\n", buffer);
  if(_from == 0)
    _arc.write_to_disk(outfile, FROM_EXTERNAL);
  else
    if(_to == 0)
      _arc.write_to_disk(outfile, TO_EXTERNAL);
    else
      _arc.write_to_disk(outfile, NO_EXTERNAL);
  sprintf(buffer, "%d", _latency);
  fprintf(outfile, "%s\n", buffer);
  sprintf(buffer, "%d\n%d\n%d", _latency_font, _latency_x,
          _latency_y);
  fprintf(outfile, "%s\n", buffer);
  if(_is_deleted)
    fprintf(outfile, "TRUE\n");
  else
```

```
    fprintf(outfile, "FALSE\n");
  if(_is_new)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(_is_modified)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(_is_state_variable)
    fprintf(outfile, "TRUE\n");
  else
    fprintf(outfile, "FALSE\n");
  if(ferror(outfile)) {
    clearerr(outfile);
    return FAILED;
  }
  else
    return SUCCEEDED;
}

//   Draws the stream.  The actual curved line is contained
// in a spline, which draws itself.

void StreamObject::draw(DRAW_STYLE style) {
  GC draw_context;
  EXTERN_STATUS status;

  if(_is_deleted == FALSE) {
    if((_from_ptr != NULL) || (_to_ptr != NULL)) {
      if(style == SOLID)
        draw_context = _graphics_context;
      else
        if(style == ERASE)
          draw_context = _erase_context;
        else
          draw_context = _dotted_context;
      if(_from_ptr == NULL)
        status = FROM_EXTERNAL;
      else
        if(_to_ptr == NULL)
          status = TO_EXTERNAL;
        else
          status = NO_EXTERNAL;
      _arc.draw(this, draw_context, _graphics_context, style, status);
      draw_text(style);
      if((_from_ptr == NULL) || (_to_ptr == NULL)) {
        if(_from_ptr == NULL)
          _arc.set_extern_location(extern_location,
                                   FROM_EXTERNAL);
        else
          if(_to_ptr == NULL)
            _arc.set_extern_location(extern_location,
                                     TO_EXTERNAL);
```

```
        set_font(draw_context, _name_font);
        XDrawString(_display_ptr, _draw_window, draw_context,
                    extern_location.x, extern_location.y,
                    "External", strlen("External"));
        XDrawString(_display_ptr, *_drawing_area_pixmap,
                    draw_context, extern_location.x,
                    extern_location.y, "External",
                    strlen("External"));
      }
    }
  }
}

void StreamObject::erase() {

  draw(ERASE);
}

//   Returns TRUE if the given coordinates are close to the
// line, or within the boundaries of the text strings.

BOOLEAN StreamObject::hit(int x, int y) {

  if(_is_deleted)
    return FALSE;
  else {
    if(_name_ptr != NULL) {
      if(strlen(_name_ptr) != 0)
        if(((x >= _name_x) && (x <= (_name_x + _name_width))) &&
           (y >= _name_y - _name_height) && (y <= _name_y)) {
          _name_selected = TRUE;
          return TRUE;
        }
    }
    if(_latency_string_ptr != NULL) {
      if(strlen(_latency_string_ptr) != 0)
        if(((x >= _latency_x) &&
            (x <= (_latency_x + _latency_width))) &&
           (y >= _latency_y - _latency_height) &&
           (y <= _latency_y)) {
          _latency_selected = TRUE;
          return TRUE;
        }
    }
    return _arc.hit(x, y);
  }
}

//   Sets the _from_ptr and _to_ptr equal to the locations of
// the from and to operators.  Simplifies getting properties
// of the operators when needed.

void StreamObject::set_object_ptrs(GraphObjectList *parent) {
```

```
  if(_from != 0)
    _from_ptr = (OperatorObject *) parent->
                          target_object(OPERATOROBJECT, _from);
  if(_to != 0)
    _to_ptr = (OperatorObject *) parent->
                           target_object(OPERATOROBJECT, _to);
  _arc.set_object_ptrs(_from_ptr, _to_ptr);
  set_default_text_location();
}

//   Notifies the stream that the given object has been deleted.
// If it's an operator connected to the stream, the stream
// deletes itself.

void StreamObject::delete_notify(CLASS_DEF class_type,
                                 int deleted_obj_id) {

  if((class_type == OPERATOROBJECT) &&
     ((_from == deleted_obj_id) || (_to == deleted_obj_id))) {
    erase();
  _is_deleted = TRUE;
  }
}

void StreamObject::replace_name(char *new_name) {

  delete _name_ptr;
  _name_ptr = strdup(new_name);
  set_text_dimensions();
}

//   Notifies the stream that the given object has moved.  If
// it's a connected object, the endpoints of the stream must
// move.

void StreamObject::move_notify(CLASS_DEF object_type, int object_id) {

  if(object_type == OPERATOROBJECT) {
    if((_from == object_id) || (_to == object_id))
      _arc.set_object_ptrs(_from_ptr, _to_ptr);
  }
  else
    if((object_type == STREAMOBJECT) && (object_id == _id))
      _arc.set_object_ptrs(_from_ptr, _to_ptr);
}

void StreamObject::select() {

  erase();
  _is_selected = TRUE;
  draw(SOLID);
}

void StreamObject::unselect() {
```

```
  erase();
  _is_selected = FALSE;
  _name_selected = FALSE;
  _latency_selected = FALSE;
  draw(SOLID);
}

//   Returns true if any of the stream's handles enclose the
// given coordinates.

BOOLEAN StreamObject::hit_handle(int x, int y) {
  EXTERN_STATUS status;

  if(_from_ptr == NULL)
    status = FROM_EXTERNAL;
  else
    if(_to_ptr == NULL)
      status = TO_EXTERNAL;
    else
      status = NO_EXTERNAL;
  return _arc.hit_handle(x, y, status);
}

void StreamObject::set_default_text_location() {

  _arc.set_text_location(_name_x, _name_y, _latency_x,
                         _latency_y);
}

void StreamObject::set_default_name_location() {

  _arc.set_name_location(_name_x, _name_y);
}

void StreamObject::set_default_latency_location() {

  _arc.set_latency_location(_name_x, _name_y, _latency_x,
                            _latency_y);
}

//   Sets latency from the value entered in the properties
// dialog box.

void StreamObject::set_constraint(int constraint) {
  char buffer[INPUT_LINE_SIZE];

  _latency = constraint;
  delete _latency_string_ptr;
  if(_latency == NULL_VALUE)
    _latency_string_ptr = NULL;
  else {
    if(_latency < 0)
      sprintf(buffer, "%d us", -_latency);
```

```
    else
      sprintf(buffer, "%d ms", _latency);
    _latency_string_ptr = strdup(buffer);
  }
  _latency_string_ptr = strdup(buffer);
  set_text_dimensions();
}

//   Sets values for the dimensions of the text strings.
//   _name_font and _met_font must be set before calling!

void StreamObject::set_text_dimensions() {

  if(_name_ptr == NULL) {
    _name_width = 0;
    _name_height = 0;
  }
  else {
    _name_width = font_text_width(_name_font, _name_ptr);
    _name_height = font_text_height(_name_font);
  }
  if(_latency != NULL_VALUE) {
    _latency_width = font_text_width(_latency_font,
                                     _latency_string_ptr);
    _latency_height = font_text_height(_latency_font);
  }
  else {
    _latency_width = 0;
    _latency_height = 0;
  }
}

void StreamObject::set_object_font(int font_id) {

  if(_name_selected)
    _name_font = font_id;
  else
    if(_latency_selected)
      _latency_font = font_id;
  set_text_dimensions();
}

//   Moves appropriate text strings the given amount.

void StreamObject::move_text(int x, int y) {

  erase_text();
  if(_name_selected) {
    _name_x += x;
    _name_y += y;
  }
  else
    if(_latency_selected) {
      _latency_x += x;
```

```
      _latency_y += y;
    }
  draw_text(SOLID);
}

//   Notifies the stream that the given object has been
// undeleted.  If it's an object connected to the stream,
// the stream may need to undelete itself.  The undeleted object
// could be the stream itself.

void StreamObject::undelete_notify(CLASS_DEF class_type,
                                   int deleted_obj_id) {

  if((class_type == STREAMOBJECT) &&
     (deleted_obj_id == _id)) {
    _is_deleted = FALSE;
    _is_modified = TRUE;
    _is_selected = FALSE;
    _name_selected = FALSE;
    _latency_selected = FALSE;
  }
  else
    if((class_type == OPERATOROBJECT) &&
       (_is_deleted == TRUE)) {
      if(_from == deleted_obj_id) {
        if(_to_ptr != NULL) {
          if(_to_ptr->is_deleted() == FALSE) {
            _is_deleted = FALSE;
            _is_modified = TRUE;
            _is_selected = FALSE;
            _name_selected = FALSE;
            _latency_selected = FALSE;
          }
        }
        else {
          _is_deleted = FALSE;
          _is_modified = TRUE;
          _is_selected = FALSE;
          _name_selected = FALSE;
          _latency_selected = FALSE;
        }
      }
      else
        if(_to == deleted_obj_id) {
          if(_from_ptr != NULL) {
            if(_from_ptr->is_deleted() == FALSE) {
              _is_deleted = FALSE;
              _is_modified = TRUE;
              _is_selected = FALSE;
              _name_selected = FALSE;
              _latency_selected = FALSE;
            }
          }
          else {
```

```
            _is_deleted = FALSE;
            _is_modified = TRUE;
            _is_selected = FALSE;
            _name_selected = FALSE;
            _latency_selected = FALSE;
          }
        }
    }
}

//   The handles_drawn states are used to prevent the handles
// from accidentally redrawing and erasing a handle.

void StreamObject::reset_handles_drawn_state() {

  _st_handles_drawn = FALSE;
  _name_handles_drawn = FALSE;
  _latency_handles_drawn = FALSE;
  _arc.reset_handles_drawn();
}

int StreamObject::text_height() {

  if(_name_selected)
    return _name_height;
  else
    if(_latency_selected)
      return _latency_height;
    else
      return 0;
}

int StreamObject::text_width() {

  if(_name_selected)
    return _name_width;
  else
    if(_latency_selected)
      return _latency_width;
    else
      return 0;
}

//   Relocates the appropriate text strings at the given
// locations.

void StreamObject::text_locate(int x, int y) {

  if(_latency_selected) {
    _latency_x = x - _latency_width / 2;
    _latency_y = y + _latency_height / 2;
  }
  else
    if(_name_selected) {
```

```
        _name_x = x - _name_width / 2;
        _name_y = y + _name_height / 2;
      }
    }
```

REFERENCES

[BERZINS91] Berzins, Valdis, and Luqi, *Software Engineering with Abstractions*, Addison-Wesley, 1991.

[CUMMINGS90] Cummings, Mary Ann, The Development of User Interface Tools for the Computer-Aided Prototyping System, M. S. Thesis, Naval Postgraduate School, Monterey, California, December 1990.

[HELLER91] Heller, Dan, *Motif Programming Manual for OSF/Motif Version 1.1*, O'Reilly and Associates, 1991.

[JOHNSON89] Johnson, Eric, and Reichard, Kevin, *X Window Applications Programming*, MIS Press, 1989.

[LUQI88-1] Luqi and M. Ketabchi, *A Computer-Aided Protytping System*, IEEE Software, March, 1988.

[LUQI88-2] Luqi, Berzins, V., and Yeh, R., *A Prototyping Language for Real-Time Software*, IEEE Transactions on Software Engineering, October 1988.

[LUQI89-1] Luqi, *Handling Timing Constraints in Rapid Prototyping*, Proceedings of the Twenty-Second Annual Hawaii Conference on Systems Science, January 1989.

[LUQI89-2] Luqi, *A Graph Model for Software Evolution*, IEEE Transactions on Software Engineering, August, 1990.

[LUQI92] Luqi, *Computer-Aided Prototyping for a Command-and-Control System Using CAPS*, IEEE Software, January 1992.

[REPS89] Reps, Thomas, and Teitelbaum, Tim, *The Synthesizer Generator: A System for Constructing Language-Based Editors*, Springer-Verlag, 1989.

[STANFORD91] Stanford University, *InterViews Reference Manual*, Version 3.0, 1991.

[VLISSIDES89] Vlissides, John, and Linton, Mark, *Unidraw: A Framework for Building Domain-Specific Graphical Editors*, Proceedings of ACM SIGGRAPH/ SIGCHI, November, 1989.

[YOURDON89] Yourdon, Edward, *Modern Structured Analysis*, Yourdon Press, 1989.

[ZYDA90] Zyda, Michael, *Book Number 5, Graphics and Video Laboratory*, Class Notes, Naval Postgraduate School, Monterey, California, 1990.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, VA 22304-6145

2. Dudley Knox Library — 2
   Code 52
   Naval Postgraduate School
   Monterey, CA 93943

3. Computer Science Department — 2
   Code CS
   Naval Postgraduate School
   Monterey, CA 93943

4. Office of the Assistant Secretary of the Navy — 1
   Research Development and Acquisition
   Department of the Navy
   Attn: Mr. Gerald A. Cann
   Washington, DC 20380-1000

5. Commander, Naval Information Systems Management Center — 1
   (Attn: RADM Moore)
   2211 Jefferson Davis Highway
   Suite 334
   Arlington, VA 22202

6. Director of Defense Information — 1
   Office of the Assistant Secretary of Defense
   (Command, Control, Communications, & Intelligence)
   Attn: Mr. Paul Strassmann
   Washington, DC 20301-0208

7. Center for Naval Analysis — 1
   4401 Ford Avenue
   Alexandria, VA 22302-0268

8. Prof. Berzins, Code CSBe — 10
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

9. Chairman, Code CS 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5100

10. Prof. Luqi, Code CSLq 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

11. Chief of Naval Research 1
800 N. Quincy Street
Arlington, VA 22217

12. Director, Ada Joint Program Office 1
OUSDRE (R&AT)
Room 3E114, The Pentagon
Attn: Dr. John P. Solomond
Washington, DC 20301-0208

13. Naval Ocean Sea Center 1
Naval Command Control and Ocean Surveillance Center,
Research Development Test and Evaluation Division
(Code 411)
(Attn: Hans Mumm)
271 Catalina Blvd.
San Diego, CA 92151-5000

14. Office of Naval Technology (ONT) 1
Code 227
Attn: Dr. Elizabeth Wald
800 N. Quincy St.
Arlington, VA 22217-5000

15. U. S. Army Institute for Research in Management 1
Information, Communications, and Computer Science
(Attn: Glenn Racine)
115 O'Keefe Blvd.
Georgia Tech
Atlanta, GA 30332-0800

16. Defense Advanced Research Projects Agency (DARPA) 1
ISTO
1400 Wilson Boulevard
Attn: LCol Eric Mattala
Arlington, VA 2209-2308

17. Assistant Secretary of the Navy for Research, Development, and 1
Acquisition (C4I, EW/Space)
(Attn: Dr. Ed Whitman)
Washington, D. C. 20350-1000

18. Attn: Dr. Charles Holland 1
Computer Science
Department of the Air Force
Bolling Air Force Base
Washington, DC 20332-6448

19. Commandant of the Marine Corps 1
Ada Joint Program Representative
Code CCI
Attn: Capt Gerald Depasquale
Washington, DC 20301

20. Department of Computer Science 1
Attn: Prof. J. F. Naveda
University of Scranton
Scranton, PA 18510

21. LCdr Don Brutzman, Code OR/BR 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

22. Regional Automated Services Center 1
Attn: Capt R. M. Dixon
Camp Lejeune, NC 28542

23. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. Gary Koob
800 N. Quincy Street
Arlington, VA 22217-5000

24. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. A. M. Van Tilborg
800 N. Quincy Street
Arlington, VA 22217-5000

25. Office of Naval Research 1
Computer Science Division, Code 1133
Attn: Dr. R. Wachter
800 N. Quincy Street
Arlington, VA 22217-5000

26. University of CA at Berkeley 1
Department of Electrical Engineering and
Computer Science
Computer Science Division
Attn: Dr. C.V. Ramamoorthy
Berkeley, CA 90024

27. Office of the Chief of Naval Operations 1
Attn: Dr. John Davis (OP-094H)
Washington, DC 20350-2000

28. United States Laboratory Command 1
Army Research Office
Attn: Dr. David Hislop
P. O. Box 12211
Research Triangle Park, NC 27709-2211

29. Hewlett Packard Research Laboratory 1
Mail Stop 321
1501 Page Mill Road
Attn: Dr. Martin Griss
Palo Alto, CA 94304

30. Persistent Data Systems 1
75 W. Chapel Ridge Road
Attn: Dr. John Nester
Pittsburgh, PA 15238

31. Commandant of the Marine Corps 1
Code TE-06
Washington, DC 20301

846-217

AMERICAN BINDERY
9-93

DUDLEY KNOX LIBRARY
3 2768 00034305 7